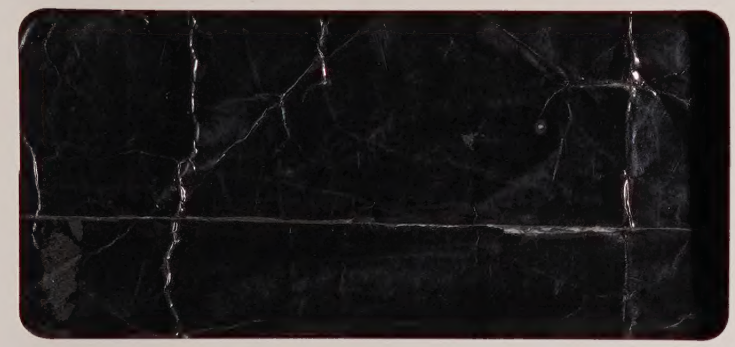
GenRad

Development Systems Division



MICROCOMPUTER SYSTEMS

Z80 DEVELOPMENT MANUAL

2302-5024-00

PRELIMINARY

February 1982

FutureData
5730 Buckingham Parkway
Culver City, CA 90230

REVISION HISTORY

Title	Number	Date	Notes
Z80 Development Manual	2302-5024-00	2/82	Preliminary Software Versions: Assembler 4.2 Linker 5.2 Debugger 2.5

RELATED PUBLICATIONS

UDOS Reference Manual	2301-5002-01
UDOS Programmer's Guide	2301-5006-00
Z80 Programming Manual V2.0 User's Manual	2300-5021-00
Z80 Micro Reference Card	2300-5022-00

© FutureData, 1982

PREFACE

This manual describes the FutureData Assembler, Linker, and Slave Emulator package, which runs on a 2300 Advanced Development System, and which is intended for the development of programs by means of the Z80 processor.

This is a reference manual, not a tutorial; it assumes familiarity with the 2300 ADS, the standard UDOS facilities, and the general concepts of at least one assembly language.

Please note that a **Documentation Reply Card** is inserted at the back of this manual. When you complete and return it, you help us produce better documentation for you.

A **User Registration Card** is included in the set of manuals you receive with your FutureData system. When you complete and return the User Registration Card, you ensure that you will receive all updates and new information for your configuration.

For your convenience, a list of **GenRad/DSD Service Locations** is appended to this manual.

ASSEMBLER

INTRODUCTION

The Assembler translates symbolic programs for the Z80 microprocessor into executable binary code. In most cases, the symbolic forms of machine instructions and operands are identical to those of Mostek's Z80 Programming Manual V2.0. Most of the Assembler directives, including macro and conditional assembly, are diffenent from those of the manufacturer. Additionally, arithmetic expressions differ somewhat in syntax.

REFERENCES

Mostek's Z80 Programming Manual V2.0 (Futuredata part number 2300-5021-00) describes the instruction set and hardware interfaces of the Z80 microprocessor in detail. Mostek's Z80 Micro Reference Card (Futuredata part number 2300-5022-00) provides a convenient reference for the Z80 instruction set.

ASSEMBLER INPUT

The Assembler accepts assembly language source code from a source file. The source may reside on a local or network diskette, a network hard disk, or a local Memory Expansion Unit. Optionally, the Assembler reads macro definitions from a macro library file. The Assembler reads and evaluates its input one line at a time.

A sample Assembler input sequence is shown in Figure 5-1.

```
Label      Opcode      Operand                 Comments
Field

*
*  Routine to copy a message
*
           SPC
           RSEG        COPYM          ; Start RSEG COPYM
           GLBL        MSGF,MSGT
           SPC
COPY       LD          HL,MSGF        ; Entry point, load "from" address
           LD          DE,MSGT        ; Load D.E with "destination" address
           LD          C,16           ; Load C register with count
           SPC
LOOP       LD          A,(HL)         ; Load A from MSGF
           LD          (DE),A         ; Store A to MSGT
           INC         HL             ; Increment MSGF pointer
           INC         DE             ; Increment MSGT pointer
           DEC         C              ; Decrement count
           JP          NZ,LOOP        ; If count not 0, jump to LOOP
           SPC
           END                        ; End of this assembly
```

Figure 1-1. Example Assembler Input

ASSEMBLER OUTPUT

The Assembler generates an output file of relocatable object code which can be processed by the Linker. The Linker combines the file with other relocatable files and assigns absolute memory addresses. A symbol table may be appended to the output file if the user wishes to reference memory addresses by way of symbolic names while debuging with the Slave Emulator.

The Assembler can also print and display a program listing containing each input line, the hexadecimal representation of the object code generated by that line, and other information.

The example input shown in **Figure 1-1** produces the listing output shown in **Figure 1-2.** The first column contains the hexadecimal addresses within the program segment. The second column contains the object output in hexadecimal.

```
Loc.     Assembler  Label  Opcode  Operand          Comments
Counter  Hex Code   Field

                    *
                    *  Routine to copy a message
                    *
0000                       RSEG    COPYM        ; Start RSEG COPYM
                           GLBL    MSGF,MSGT

0000     210000     COPY    LD      HL,MSGF      ; Entry point, load "from"
                                                   addr
0003     110000             LD      DE,MSGT      ; Load D,E with "destination"
                                                   addr
0006     0E10               LD      C,16         ; Load C register with length
                                                   count
0008     7E         LOOP    LD      A,(HL)       ; Load A from MSGF
0009     12                 LD      (DE),A       ; Store A to MSGT
000A     23                 INC     HL           ; Increment MSGF pointer
000B     13                 INC     DE           ; Increment MSGT pointer
000C     0D                 DEC     C            ; Decrement count
000D     C20800             JP      NZ,LOOP      ; If count not 0, jump to LOOP

0010                       END                  ; End of this assembly
```

Figure 1-2. Example Assembler Output

For each assembly, a symbol table listing is produced showing the memory address of each global (G) and local symbolic label name and each external reference (X) (see **Figure 1-3**).

```
======= Z80 ASSEMBLER V4.2 ================================== FutureData =====


  COPY     0000     COPYM     G 0000     LOOP      0008     MSGF      X 0000
  MSGT   X 0000
```

Figure 1-3. Example Symbol Table Listing

PROGRAM SEGMENTS

Assembly language programs are divided into program segments. A program segment A program segment may be either absolute or relocatable. The address at which an absolute segment is loaded is specified at assembly time. The address at which a relocatable segment is loaded is determined later by the Linker.

There can be only one absolute segment (ASEG) per assembly - it is unnamed. An assembly can contain as many as eight relocatable segments (RSEG). Each RSEG can be relocated by the Linker to any location in the address space of the Z80 microprocessor.

LOCATION COUNTERS

The location counter points to the next available memory location in the current program segment. The counter is a 16-bit quantity that is incremented by one for each byte of object code output. Each assembly contains a separate counter for each RSEG and for the ASEG. The 16-bit location counter limits the size of the program segment to 65535 bytes. Larger programs can be constructed by linking together several relocatable segments from one or more assemblies.

ASSEMBLER OPTIONS

Upon entry to the Assembler, a list of options appears on the display screen. Options may be selected in any order by typing the appropriate letter(s) as listed in Table 1-1.

Table 1-1. Assembler Options

Option	Description
L	Display a program listing.
T	Truncate lines of the display to 80 characters and limit printer or display listing of the DC directive to one line. If this option is not specified, all lines generated by the DC directive will be output one byte per line.
E	Display only lines containing errors flagged by the Assembler for display or printing.
S	Append the table of symbolic address labels to the end of the relocatable object file.
Z	Flag all lines containing non-8080 instructions (this option is used with the Z80 Assembler only). Non-8080 lines are flagged with the message: INVALID OPCODE. The Z option is useful for verifying Z80 and 8080 program compatibility.

SPECIFYING ASSEMBLER FILES

After the options have been selected and entered, the Assembler generates a series of prompts, as listed in **Figure 1-4.** The user can respond to each with any legal file specification.

1) SOURCE FILE - Any filename entered must have the S attribute. This input is required.

2) MACRO FILE - This input is optional (enter <RETURN> to bypass).

3) OUTPUT FILE - This input is optional (enter <RETURN> to bypass). Any filename entered must have the R attribute (a new file created by the Assembler will automatically be assigned the R attribute).

4) LISTING FILE - This input is optional (enter <RETURN> to bypass). Any filename entered must have the S attribute or none at all.

Figure 1-4. Example Assembler Display

ASSEMBLER PROCESSING

After the last filename is entered and <RETURN> is pressed, assembly begins. Statements are processed in two passes. On the first pass the Assembler assigns values to symbolic labels and stores them in a symbol table. During pass 2 the Assembler analyzes each statement from the input file and generates relocatable object output. If the Assembler recognizes an error, the statement is flagged and zero bytes are generated for that statement.

HALT ASSEMBLY

To halt the assembly process at any point, press <RETURN>. This is useful for viewing the Assembler output as it scrolls on the display. The Assembler pauses and prompts:

```
CONTINUE?
```

To continue, type:

```
Y or <RETURN>
```

To abort assembly, type:

```
N or <BREAK>
```

ASSEMBLER STATEMENT SYNTAX

Statements are composed of four components or fields. Fields are separated by a space or a series of spaces. Statements are entered one per line with a maximum of 80 characters.

Each line input to the Assembler must fall into one of the following categories:

1) An assembly language instruction is of the form:

   [label][:] mnemonic [operand[,operand]...]] [;comment]

   The label is optional and is a name assigned by the programmer. The mnemonic field contains a Z80 instruction mnemonic. The presence and number of operand fields vary according to the instruction mnemonic. Each operand specifies an effective address (machine register, memory reference, or immediate data), or other instruction-specific information. The optional comment field contains explanatory remarks. It is not actually assembled. Each instruction causes the Assembler to output one or more words (two bytes per word) of object code.

2) An Assembler directive. There are two types of Assembler directives:

   a) those which cause the Assembler to perform the clerical tasks of assembly;

   b) macro directives and conditional assembly directives.

3) A comment line. The Assembler treats all characters to the right of a semicolon, or any line with a semicolon or asterisk in the first character position, as a comment. An entirely blank line is also processed as a comment.

LABEL FIELD

The optional label field may contain a symbolic name which is used to reference the statement or which will be assigned a value by an EQU directive. Labels must begin in the first character position in the line, and labels may be from one to eight characters in length. The first character must be a letter (A-Z) or a dollar sign ($). Subsequent characters must be letters, dollar signs or numbers (0-9).

During assembly a value is assigned to each label, as described below.

1) Absolute — In absolute segemnts, the value assigned to a label is the absolute address at which the assembled statement is stored. This value is identical to the value in the absolute segment location counter, which appears on the left side of the assembly listing.

2) Relocatable — In RSEGs, the value assigned to a label is the local address within the segment. This value is identical to the value in the location counter of that segment, which appears on the left side of the assembly listing. When the segment becomes part of an executable program during Linker processing, the Linker adds the absolute address of the first byte in the segment to the local address of the label. The result is the absolute address assigned to the label.

3) Equated — When the EQU Assembler directive is encountered, the label is assigned the value of the operand field. Further information on Assembler directives is given later in this chapter.

The Assembler writes each label name, the local address of the label, and other information into a symbol table. Label names are represented in ASCII, with each character requiring one byte of memory.

OPERATION CODE FIELD

The operation code field contains an instruction mnemonic assigned by the microprocessor manufacturer, an Assembler directive, or a macro name. This field must begin in the second character position on the line or thereafter, and cannot be more than six characters in length. Each instruction causes the Assembler to generate zero or more bytes of relocatable object code. An operation code must be present in all assembly input lines, except in comment lines.

OPERAND FIELD

The operand field contains additional parameters (if any) associated with the instruction or directive in the opcode field. Operands are separated by commas. The operand field begins after the opcode field, but the two fields must be separated by at least one space. A space terminates the operand field. If a space is inadvertently included in the field, the portion to the right of the space will be read as a comment.

The content of the operand field depends upon each instruction or directive. The operand of directives are usually expressions. The operands of Z80 instructions are usually effective address operands.

COMMENT FIELD

The comment field allows the programmer to document the action or purpose of a statement. This field is not processed by the Assembler, but is printed or displayed in the program listing. The first character of a comment field must be a semicolon. The semicolon terminates Assembler processing of the line in which it appears. The semicolon and those characters to the right of the semicolon are treated as a comment. The comment field may begin after the operand field if the two fields are separated by at least one space.

EXPRESSIONS

Expressions compute or stand for strings, numbers, and addresses. Expressions consist of terms, which are values, and operators which alter or combine values. Expressions are evaluated from left to right. Each operator has a precedence, which controls the order of evaluation. Parentheses may be used to specify an order of evaluation different from that implied by the operator precedence.

EXPRESSION TERMS

Each term stands for a value. There are seven types of terms:

1) Decimal Constants: Decimal constants are numbers containing digits ranging from 0 to 9. Single-byte decimal constants may range in value from 0 to 255; two-byte decimal constants may range in value from 0 to 65,535.

2) Hexadecimal Constants: Hexadecimal constants consist of the digits 0 to 9 and the letters A to F. The digits must be enclosed in single quotes, and an X must prefix the first quote. Single-byte constants may be one or two hexadecimal digits. Two-byte (memory address) constants may be from one to four hexadecimal digits in length. The constant is treated as if leading zeros were supplied when less than the maximum number of digits is used.

   Examples of single-byte hexadecimal constants are show below:

```
X'1' (same as X'01')
 X'A3'
```

   Examples of two-byte (address) hexadecimal constants are shown below:

```
X'103' (same as X'0103')
 X'BF2A'
```

3) ASCII Character Constants: A character constant consists of one or two characters enclosed in single quotes. Each character is given an eight-bit (one-byte) value. A string of three or more ASCII characters may be used as an operand only with relational operators. If one operand of a relational operator is a string and one is another type of operand, the string is always greater. If both operands of a relational operator are character strings, comparison is done on a byte-by-byte basis using the ASCII collating sequence for determining the comparison results.

Examples:

'AB' is assembled to X'4142'
' ' (space) is assembled to X'20'

4) Symbolic Labels: During Assembler pass 1 each statement label is stored in the symbol table along with the location counter value. During pass 2, the value assigned to a label is substituted wherever it occurs as an operand.

5) Asterisks: An asterisk used as an operand causes the Assembler to use the location counter value of the first byte of the statement in which the asterisk occurs.

Expressions are evaluated from left to right, resulting in a 16-bit (two-byte) quantity. However, the Assembler directive or microprocessor instruction (in which the expression appears as an operand) may truncate the expression to eight bits or may reverse the order of the bytes.

All expressions must reduce to one of the following forms:

a) an absolute value;

b) a relocatable value ± an absolute value;

c) an external value ± an absolute value.

There is one exception. An absolute value expression may contain a pair of relocatable terms if the terms are in the same segment and if, when subtracted, the result is greater than zero. Relocatable terms cannot be added.

The individual operands (terms) in an expression are connected by any of the following types of operators: arithmetic, logical, register shift, relational, byte-extraction or byte-order.

1) Arithmetic Operators

+	Addition.
-	Subtraction or unary minus sign.
*	Multiplication.
/	Integer division. Any remainder is truncated.
n.MOD.m	Modulo (remainder). The absolute value of the left term is divided by the absolute value of the right term. The result is the remainder from the division, with the sign of the result being negative if the left term was negative.

2) Logical Operators

.AND.	Performs bit-by-bit logical AND.
.OR.	Performs bit-by-bit logical OR.
.NOT.	Complements each bit of the following one-byte operand or the low-order byte of a two-byte operand.
.XOR.	Performs bit-by-bit exclusive-OR operation (a or b, but not both).

3) Register Shift Operators

.SHL. Shifts the left term left by the number of bits specified by the right operand. The vacated low order bit positions of the result are set to zeros.

.SHR. Shifts the left term right by the number of bits specified by the right operand. This is an arithmetic shift, with the highest-order (sign) bit of the left term being propagated into the vacated high-order bit positions of the result.

4) Relational Operators

All relational Operators evaluate to a number value which is zero if the relation is false, or X'FFFF' if the relation is true.

= Equal To operator. True if left term is equal to right term.

<> Not Equal To operator. True if left term is not equal to right term.

> Greater Than operator. True if left term is greater than right term.

< Less Than operator. True if left term is less than right term.

>= Greater Than Or Equal To operator. True if left term is greater than or equal to right term.

<= Less Than Or Equal To operator. True if left term is less than or equal to right term.

A signed comparison is performed for numbers. Thus, in 16-bit two's complement notation:

```
-1          is equal to     X'FFFF'
-1          is less than    1
X'FFFF'     is less than    1
```

The following rules govern the use of ASCII character strings with relational operators:

a) If one term is a one or two character ASCII string and the other term is a number, the string is taken as a number and compared to the number term.

b) If one term is an ASCII string of more than two characters and the other term is a number, the string is considered to be greater than the number.

c) If both terms are ASCII strings, the comparison is performed on a character-by-character basis according to the ASCII collating sequence. If one string is shorter than the other, and the two strings are identical for the length of the shorter string, the shorter string is considered to be less than the longer string.

Examples:

```
'ABC'   is greater than   'AAA'
'ABC'   is less than      'ABCDEF'
'ABC'   is greater than   'ABATE'
```

5) Byte-Extraction Operators

H(expression) Extracts the high-order byte.

L(expression) Extracts the low-order byte.

The high- or low-order eight bits of an expression may be extracted by enclosing the operand in parentheses and prefixing it with H (high) or L (low). These operators are used when an expression having a 16-bit value must be truncated to an eight-bit value. For example:

H(X'2E35'-2) Evaluates to X'2E'.

L(X'2E35'+2) Evaluates to X'37'.

H(ALPHA) evaluates to the high-order eight bits of the value assigned to the symbolic label ALPHA. Byte extraction is not allowed as a term of an expression.

6) Byte-Order Operators

A(expression) High-order byte is stored first.

B(expression) Low-order byte is stored first.

The byte order of the 16-bit result of an expression evaluation may be specified by enclosing the expression in parentheses and prefixing it with the letter A or B. If a byte-order operand is not used, the high order is stored first.

A byte-order operator in the operand field of a Define Constant (DC) directive causes that directive to store both bytes of the expression it encloses. A byte-order operator may not be used as a term of an expression.

PRECEDENCE OF OPERATORS

The order of evaluation of an expression is left to right. Operators with higher precedence are evaluated first, unless otherwise overridden by parentheses. The precedence of operators is shown below in decreasing order:

```
6)  *      /      .MOD.      .SHL.      .SHR.
5)  +      -  (unary and binary)
4)  =      >      <      >=      <=      <>
3)  .NOT.
2)  .AND.
1)  .OR.     .XOR.
```

COMMENT STATEMENT

If a semicolon or an asterisk is in the first character position on a line, the entire line is treated as a comment. Comment statements are useful for documentation requiring more space than is available in a comment field and for lengthy descriptions such as a program function overview. A line containing a semicolon as the first non-space character is processed as a comment statement. In a macro library file, comment statements which are not inside macro definitions terminate Assembler processing of the file.

ASSEMBLER DIRECTIVES

The following types of Assembler directives are available:

SEGMENT DIRECTIVE

Either an ASEG or an RSEG directive may appear in an assembly, preceding statements which generate object output. If segment directives do not appear, statements will be assembled into the ASEG.

The ASEG directive defines the beginning of the absolute program segment. Neither a label nor operands are allowed in the directive. The Assembler begins the ASEG at location zero or at a location defined in an immediately following ORG directive. By writing another ASEG directive, it is possible to assemble into the ASEG after intervening segments. An ASEG ends at the beginning of another segment or at the end of an assembly.

The RSEG directive defines a relocatable program segment. A maximum of eight RSEGs are allowed in an assembly, each identified by a user-supplied name in the operand field. A label is not allowed. An RSEG directive is entered as:

```
RSEG rseg-name
```

An RSEG ends at the beginning of another segment or at the end of an assembly. rseg-name is automatically made a global symbol. It is possible to assemble into a segment after intervening segments by writing an ASEG directive or an RSEG directive with the appropriate rseg-name.

The Linker and Slave Emulator use different terminology when referring to relocatable and absolute program segments; therefore, in Chapters 2 and 3 of this manual, RSEG is spelled CSECT and ASEG is spelled ASECT.

SEGMENT-ORIGIN DIRECTIVE

A segment-origin directive is entered as:

```
ORG expression
```

The origin directive specifies the segment location at which subsequent statements are assembled and sets or resets the location counter of the program segment in which it occurs. The expression must be absolute if in an ASEG, or relocatable if in an RSEG. The ORG directive may appear anywhere in a program segment. If the ORG directive is not used, the segment begins with the location counter set to zero. The location counter is incremented by one as each byte of object code is generated. Symbols appearing in the operand field expression must have been previously defined.

GLOBAL DIRECTIVE

A global directive is entered as:

```
GLBL symbol [,symbol]...
```

A global symbolic reference refers to a common memory location for other assemblies as well as the current assembly. The assembly in which a global symbol is defined and each assembly referencing the global symbol must declare that symbol to be global, using a GLBL directive. In the Assembler symbol table listing, global symbols appearing in the label field of the current assembly are flagged with the letter G. Global symbols used in operand fields are known as external symbols. They are defined outside the current assembly and are flagged with the letter X. All other listed symbols are local.

A maximum of 255 global labels are allowed in one assembly. Relocatable segment names and the entry points specified in the END directive are automatically made global and may not appear in a GLBL directive.

END DIRECTIVE

An END directive is entered as:

```
END [expression]
```

The END directive terminates Assembler operation and, optionally, specifies an entry point at which program execution may begin. If a symbolic label name appears in the expression, it automatically becomes global. The END directive is the last statement processed in an assembly input file. If the END directive is missing, an error is flagged.

EQU DIRECTIVE

An EQU directive is entered as:

```
label EQU expression
```

The EQU directive assigns the value of the expression in the operand field to a symbolic name in the label field. The expression is evaluated during Assembler pass 1. The symbol name may be referenced in the operand fields of statements occuring before and after the EQU directive. However, symbols in the operand field expression must have been previously defined. If a symbol in the operand field has not been previously defined, the line will be flagged with the message: INVALID FORWARD REFERENCE.

DEFINE-CONSTANTS DIRECTIVE

A Define-Constants (DC) directive is entered as:

```
[label]DC expression [,expression]
```

The DC directive places data in memory beginning at the current value of the location counter. Multiple operands are stored in successive memory locations. The DC directive is the only directive that generates object output.

Data are stored in five ways which may be combined in one DC statement:

1) The 16-bit value of an expression is truncated to the low eight bits and stored in one byte of memory.

2) The 16-bit value of an expression is truncated to either the low-order or high-order eight bits by a byte-extraction operator and stored in one byte of memory.

3) The 16-bit value of an expression prefixed with a byte-order operator is stored in two succeeding bytes. Byte order is determined by the operator.

4) An ASCII character string is stored one byte per character. The string is enclosed in single quotes. A single quote within the string is represented by two successive single quotes.

5) Decimal integers from 0 to 255 (hexadecimal 0 to FF) are stored in one byte. Decimal integers from 256 to 65,535 (X'100' to X'FFFF') are truncated to the lower-order eight bits.

The following example demonstrates each type of data storage:

```
LOC    DC    JUMP,L(R+8),B(LOOP),'ABC',12,A(X'A030')
```

Ten data bytes are stored in memory beginning at location LOC. The first byte stored is the low-order byte of the 16-bit memory address JUMP. The next byte stored is the low-order byte of the 16-bit result obtained when eight is added to memory address R. Then, both bytes of the 16-bit memory address LOOP are stored. Byte-order operator B causes the low-order byte to be stored first. The ASCII representation of the letters ABC is stored in the next three bytes. The decimal integer 12 is stored in the next byte. Then, the hexadecimal number A030 is stored in two successive bytes. Byte-order operator A causes the high-order byte to be stored first.

DEFINE MEMORY SPACE DIRECTIVE

A Define Memory Space (DS) directive is entered as:

```
[label]DS expression
```

The DS directive reserves memory space by advancing the segment location counter. The amount of space desired (in bytes) is entered in the operand expression. If a symbol name is used as the expression or a term of the expression, it must have been previously defined; if not defined, the line will be flagged with the message: INVALID FORWARD REFERENCE.

PRINTER CONTROL DIRECTIVES

1) SPC — This directive places a blank line in the listing for improved readability. No operands are processed.

2) EJE — The eject directive causes the Assembler to skip to the top of the next page of the listing for improved readability. No operands are processed.

3) PRINT OFF — PRINT OFF suppresses printing of all following source lines. The directive itself is not printed.

4) PRINT ON — PRINT ON lists all source lines except:

   a) lines generated in a macro expansion;
   b) lines skipped due to conditional assembly directives;
   c) conditional directives themselves.

   Lines are printed after set-symbol substitution has taken place.

5) PRINT GEN — PRINT GEN lists macro expansion lines as well as the lines listed by PRINT ON. Macro expansion lines are printed after parameter substitution and set symbol substitution have taken place. This directive affects printing and the display listing identically.

6) PRINT ALL — PRINT ALL prints all source lines including lines skipped due to conditional assembly directives and the conditional assembly directives themselves. Location counter values and object data for skipped lines are not printed. Lines are printed after set symbol and macro parameter substitutions have taken place. This directive affects printing and the display listing identically.

MACROS

The macro facility enhances Assembler capability by allowing a single calling statement to generate groups of instructions. As shown in Figure 1-5, a macro definition begins with a "MACRO" pseudo-op and ends with an "ENDM" pseudo-op.

Since it is more common to generate a group of similar rather than identical instructions, the macro statements may contain parameter names which are referenced and replaced by actual values. These parameter names are listed (separated by commas) in the operand field of the MACRO pseudo-op. Each macro parameter name may be referenced within the body of a macro definition by a statement containing the macro parameter name preceded by an ampersand. When a calling statement is encountered, the instructions contained in the macro definition are assembled and the parameter values specified in the calling statement are substituted for each ampersand and name.

```
Macro name      MACRO     parameter1,parameter2      (MACRO pseudo-op)
                 statement
                 statement & parameter1             (macro body)
                   .
                   .
                   .
                 ENDM                               (ENDM pseudo-op)
                   .
                   .
                   .
                 Macro-name                         (calling statement)
```

Figure 1-5. MACRO Definition

An example of macro operation is shown below:

```
                TEST    MACRO     ABC          (MARCO pseudo-op)
Macro                   LD        HL,&ABC      (macro parameter reference)
Definition              ENDM                   (ENDM pseudo-op)


                        TEST      'AB'         (calling statement)
Macro           +       LD        HL,'AB'      (statement generated when
Expansion                                       macro is assembled)
```

SYNTAX

A macro name must begin in the first character position of the MACRO pseudo-op. Macro names and macro parameter names (in MACRO pseudo-ops) must begin with a letter (A-Z) or a dollar sign ($). Subsequent characters must be letters (A-Z), numbers (0-9) or dollar signs. Macro names and macro parameter names may be from one to eight characters in length. Commas and blanks may exist in a parameter if placed between single quotes (there must be an even number of quotes in any parameter). Examples of valid parameters are listed below:

```
20
X'40'
'A,B'+'B,C'
```

Macro parameters may be from 0 to 32 characters in length, and may contain any character.

SUBSTITUTION

Substitution may occur in the label field, opcode field, operand field, comment field or any combination of these fields. For the example macro definition:

```
SAVEA   MACRO   TYPE,LOC1,LOC2
        &TYPE   &LOC1,&LOC2
        ENDM
```

the call:

```
SAVEA   LD,(VAL),A
```

results in:

```
LD      (VAL),A
```

the call:

```
SAVEA   LD,(DE),A
```

results in:

```
LD      (DE),A
```

LITERAL AMPERSAND

As previously stated, the ampersand signals the macro processor to substitute a parameter value. At times, however, a literal ampersand character is needed. Four consecutive ampersands are used to tell the macro processor that rather than substituting, an ampersand character is desired. In the example below, a single ampersand is stored in a constant.

```
DC      'ABC&&&&XYZ'
```

PLACEMENT OF MACRO DEFINITIONS

Macros must be defined at the beginning of the input source file or macro library file. The Assembler reads macros until it reaches a statement outside a macro definition other than PRINT, SPACE or EJE printer control directives. By specifying a macro file to the Assembler, a second source file of macro definitions (a macro library) may be included in the assembly. Macros may be defined in either the input source file, or a macro library file, or both.

DUPLICATE MACRO DEFINITIONS

If a macro library is specified, the Assembler reads macro definitions from the library before reading the main source input file. Macros in the main source file take precedence over macros in the macro library. If two macros with the same name are defined, the Assembler uses the last one read.

GENERATING UNIQUE LABELS

A macro call may generate statements containing labels. For example, the following definition is used to set the DE registers to the absolute value of the DE registers' contents (Z80 microprocessor):

```
ABSD        MACRO
            LD          A,D
            OR          A
            JP          ENDABSD
            CPL
            LD          D,A
            LD          A,E
            CPL
            LD          E,A
            INC         DE
ENDABSD     EQU         *
            ENDM
```

The first time ABSD is called, label ENDABSD is defined. Since ENDABSD has been previously defined, a duplicate label definition error occurs the second time ABSD is called. To avoid duplication, a unique label must be generated each time the macro is called. The Assembler provides a special predefined macro parameter (INDX) which is set to a unique five-digit numeric value each time a macro is called. The macro ABSD may now be defined using INDX as shown below:

```
ABSD        MACRO
            LD          A,D
            OR          A
            JP          AB&INDX
            CPL
            LD          D,A
            LD          A,E
            CPL
            LD          E,A
            INC         DE
AB&INDX     EQU         *
            ENDM
```

Assuming that this is the only macro defined and that there are exactly two calls made, the first call defines the label AB00001, and the second call defines the label AB00002.

CONCATENATING PARAMETERS

In the previous macro example, the value of parameter INDX was appended to the right of the characters AB by writing the parameter name (preceded by an ampersand) to the immediate right of the characters AB: AB&INDX. If the user wishes also to append the letter A to the right of the INDX parameter value, a problem will arise:

```
JP      AB&INDXA
```

The macro processor interprets this to mean: substitute the value for parameter INDXA. To solve this problem, the macro processor recognizes the exclamation mark (!) as the parameter name delimiter indicating the end of a parameter name. The letter A may be appended to the right of parameter INDX as shown below:

```
JP      AB&INDX!A
```

On the first call this statement becomes:

```
JP      AB00001A
```

Note that if ! appears anywhere other than after a macro parameter (or a set symbol), it will be processed as a normal character.

MACRO CALLS WITHIN MACROS

A macro definition may include a statement which calls another (or the same) macro. These calls may be nested to a level of 127 (depending on the Assembler symbol table space available). A macro definition may not have another macro defined in its body.

EXITM STATEMENT

When encountered during a macro expansion, the EXITM statement immediately terminates assembly of the current or named macro. The next statement processed will be the first one following the macro call. The correct syntax is shown below. Note that a label is not allowed.

```
EXITM macro-name
```

CONDITIONAL ASSEMBLY

Conditional assembly statements allow the programmer to selectively assemble statements in a source file. The variables used in conditional statements are known as set symbols. A set symbol value may be a number from 0 to 65,535 or a character string containing from 0 to 32 characters. Values are set by use of the DEFL (define local) and DEFG (define global) statements.

DEFL STATEMENT

This statement is used to define (and redefine) local set symbols. Local set symbols are known only in the macro in which they are defined. A set symbol of the same name defined in another macro is considered to be a different set symbol. The proper syntax for a DEFL statement is shown below:

```
     set symbol name  DEFL  expression
or
     set symbol name  DEFL  'string'
```

If apostrophes are placed around the operand, it is processed as a character string. If apostrophes are not placed around the operand, it is processed as an integer expression. Single apostrophes within strings must be doubled.

Labels may appear in the expression only if they have been previously defined. When defining a set symbol, an ampersand must not precede the name in the label field. However, each set symbol reference must be preceded by an ampersand.

DEFG STATEMENT

This statement is used to define (and redefine) global set symbols. A global set symbol is known in the entire assembly, including all macros, unless a local set symbol of the same name is defined in a particular macro. In this case, after the local set symbol is defined, the global set symbol is unknown in that macro call. The appropriate syntax for a DEFG statement is shown below:

```
     set symbol name  DEFG  expression
or
     set symbol name  DEFG  'string'
```

If apostrophes are placed around the operand, it is processed as a character string. If quotes are not placed around the operand, it is processed as an integer expression. Quotes embedded in the string must be doubled. A label may appear in the expression only if it has been previously defined.

When defining a set symbol, an ampersand must not precede the name in the label field. However, each set symbol reference must be preceded by an ampersand.

SUBSTR STATEMENT

The SUBSTR (substring) statement assigns part of a string to a set symbol. If the set symbol has not been previously defined, a new local set symbol will be defined. The proper syntax is shown below:

```
set symbol name  SUBSTR  expression A , expression B , 'string'
```

expression A defines the beginning character position of the substring; the first character is position 1. expression B defines the length of the substring. If expression B is zero, the substring will begin with the character defined by expression A and continue to the end of the string.

LENGTH STATEMENT

The LENGTH statement assigns the length of a string to a set symbol. If the set symbol has not been previously defined, a new local set symbol will be defined. The proper syntax is shown below.

```
set symbol name  LENGTH  'string'
```

IF BLOCK

During Assembler operation, an IF block selects specific sections of code for processing. IF blocks may be nested within IF blocks or DO blocks to any level. An IF block must begin with an IF statement and must end with an ENDIF statement. These two statements and the optional ELSEIF, ELSE and EXITIF statements are described below:

1) IF Statement: The IF statement begins an IF block. An optional IF statement name may be entered in the label field for referencing by the EXITIF statement. If the expression contained in the IF statement operand field is non-zero, the statements following the IF statement and preceding the first ELSE or EXITIF statement (at the same nesting level) will be processed. If the expression is zero, the statements following the IF and preceding the first ELSEIF, ELSE or ENDIF statement (at the same nesting level) will be ignored. The proper syntax is shown below:

```
[if block name] IF expression
```

2) ELSEIF Statement: The ELSEIF statement is used in conjunction with an IF statement to test an alternate condition without going to a deeper nesting level. If the expression in the IF statement is zero, the expressions in all previous ELSEIF statements at this nesting level are zero, and the expression in this ELSEIF, ELSE or ENDIF statements (at this nesting level) will be processed. The proper syntax is shown below. Note that a label is not allowed.

```
ELSEIF expression
```

3) ELSE Statement: The ELSE statement is used in conjunction with an IF statement to indicate the last alternative. The statement is identical to:

```
ELSEIF 1
```

That is, if the expressions in the IF statement and all subsequent ELSEIF statements at this nesting level are zero, the statements after the ELSE statement and preceding the closing ENDIF statement are processed. The correct syntax is shown below. Note that a label and operands are not allowed.

```
ELSE
```

4) EXITIF Statement: The EXITIF statement causes all statements preceding the closing ENDIF statement in the named or current IF block to be ignored. The correct syntax is shown below. Note that a label is not allowed.

```
EXITIF [if block name]
```

5) ENDIF Statement: The ENDIF statement terminates an IF block. The correct syntax is shown below. Note that a label and operands are not allowed.

```
ENDIF
```

Example: The ADDX macro defined earlier can be expanded to add a number between 0 and 255 to the BC, DE or HL registers as shown on the following page.

```
ADDXY       MACRO     REG,NUM
            IF        '&REG'='BC'
REGH        DEFL      'B'
REGL        DEFL      'C'
            ELSEIF    '&REG'='DE'
REGH        DEFL      'D'
REGL        DEFL      'E'
            ELSE
REGH        DEFL      'H'
REGL        DEFL      'L'
            ENDIF
            LD        A,&REGL
            ADD       A,&NUM
            LD        &REGL,A
            LD        A,&REGH
            ADC       A,0
            LD        &REGH,A
            ENDM
```

DO BLOCK

During Assembler operation, the statements contained in a DO block are repeatedly assembled. DO blocks may be nested within IF blocks or DO blocks to any level. A DO block must begin with a DO statement and end with an ENDDO statement. These two statements and the optional EXITDO and NEXTDO statements are described below.

1) DO Statement: The DO statement begins a DO block. An optional name may be contained in the label field for referencing by the EXITDO and NEXTDO statements. The expression in the operand field is evaluated once at the entry to the DO block and is stored as the DO count (the number of times the statements within the block are processed). The proper syntax is shown below:

   [do block name] DO [expression]

   If expression is omitted, the block will be processed 65,536 times (essentially, indefinitely).

2) EXITDO Statement: The EXITDO statement causes the Assembler to immediately terminate processing statements in the current or named DO block and begin at the first statement after the closing ENDDO statement. The correct syntax is shown below. Note that a label is not allowed.

   EXITDO [do block name]

3) NEXTDO Statement: The NEXTDO statement causes the Assembler to immediately begin processing the next iteration of the current or named DO block. If this is not the last iteration, the next statement processed will be the first statement after the DO statement. The correct syntax is shown below. Note that a label is not allowed.

```
NEXTDO [do block name]
```

4) ENDDO Statement: The ENDDO statement terminates a DO block. The correct syntax is as follows. Note that a label and operands are not allowed.

```
ENDDO
```

Example: One use of conditional statements is that of generating tables that would otherwise be tediously entered from the keyboard. The example below generates a table to check for a numeric character.

```
TABLE    EQU     *
CHAR     DEFG    0
         DO      256
         IF      &CHAR>='0'.OR.&CHAR<='9'
         DC      0
         ELSE
         DC      X'FF'
         ENDIF
CHAR     DEFG    &CHAR+1
         ENDDO
```

ADVANCED FEATURES

During assembly, each source line is scanned twice for ampersands which signal set symbol or macro parameter substitution. During each scan the values of the set symbol names (preceded by ampersands) are substituted for the ampersand and name. Double ampersands are replaced by a single ampersand. Two special types of set symbol specifications are allowed.

SUBSCRIPTED SET SYMBOLS

The following statement will define a set symbol whose name is based on the value of another set symbol:

```
A&I     DEFL     'A'
```

If the value of &I is the integer 4, the set symbol A00004 will be defined to have the value A.

The following statement uses the generated set symbol:

```
     LD      B,'&&A&I'
```

On the first scan, && is replaced by &, and &I is replaced by 00004, resulting in:

```
     LD      B,'&A00004'
```

On the second scan, &A00004 is replaced by A, resulting in:

```
     LD      B,'A'
```

By varying the value of set symbol I, set symbols may be defined, referenced and indexed (using I as a subscript).

INDIRECT SET SYMBOLS

The following statements define a set symbol whose value is the name of another set symbol:

```
     A     DEFG     'ABC'
     B     DEFG     'A'
```

The following statement uses set symbol B as an indirect reference to the value of set symbol A:

```
     IF     '&&&B'='ABC'
```

On the first scan, the first two ampersands are replaced by a single ampersand, and &B is replaced by A, resulting in:

```
     IF     '&A='ABC'
```

on the second scan, &A is replaced by ABC, resulting in:

```
     IF     'ABC'='ABC'
```

LIST OF ASSEMBLER MESSAGES

DUPLICATE SYMBOL

Respecify.

ERROR WHILE WRITING OBJECT FILE

Terminates Assembler processing.

EXPRESSION HAS MORE THAN 1 RELOCATABLE FACTOR

Expressions with relocatable or absolute terms.

IMMEDIATE VALUE > -128 or < 127

The instruction allows only a signed 8-bit value. The high-order bit is used as the sign. Respecify.

INVALID FORWARD REFERENCE

Example: A EQU B
where B is not yet defined.

INVALID LABEL

Respecify.

INVALID OPCODE

Respecify.

INVALID OPERAND

Respecify.

MISSING END

An END statement must be the last statement in an assembly.

MORE UNPRINTABLE ERRORS

Occurs if there are more than 10 errors in one line.

OVER 255 EXTERNALS

Terminates Assembler processing.

RELATIVE JUMP OUT OF RANGE

Appropriate processors only.

RELATIVE JUMP TO DIFFERENT RSEG

Appropriate processors only.

RELATIVE JUMP TO EXTERNAL SYMBOL

Appropriate processors only.

SYMBOL TABLE OVERFLOW

Terminates Assembler processing.

UNDEFINED SYMBOL

Respecify.

LIST OF MACRO AND CONDITIONAL ASSEMBLER MESSAGES

ELSE OUT OF PLACE

ENDDO OUT OF PLACE

ENDIF OUT OF PLACE

ENDM OUT OF PLACE

EXITM OUT OF PLACE

INVALID MACRO PARAMETER NAME

MACRO DEFINITION OUT OF PLACE

MACRO NESTING EXCEEDS 127

MISSING ENDDO

MISSING ENDIF

MISSING ENDM

OPERAND LONGER THAN 32 CHARS

UNDEFINED SET SYMBOL

LINKER

INTRODUCTION

The Linker combines relocatable program sections (CSECTs) from Assembler- or Compiler-generated relocatable file(s) to form a single executable absolute object file. Address references between sections that were unresolved at assembly or compile time are resolved, and sections are relocated for loading at absolute addresses. In addition to relocatable sections, the Linker also supports an absolute section, the special program section for which no relocation is necessary.

LINKER INPUT

Assembler- or Compiler-generated relocatable object files may be used as input to the Linker. These object files are distinguished by the "R" attribute. Each relocatable object file may contain a maximum of eight relocatable sections and one absolute section. If two or more input files contain the same CSECT name, the Linker will either:

1) Ignore the second through nth definitions of that CSECT.

2) If the "append" mode is in effect, all CSECTs of the same name are concatenated into a single memory area by the Linker.

If two or more input files contain the ASECT directive, memory allocations may overlap. If CSECT directives are not used in an assembly, object code will be placed in the ASECT.

In order to link symbolic name references in object files, the name must be declared public (using the PUBLIC directive) during the assembly or compilation of the module which defines the name and external (using the EXTRN directive) in any modules which refer to the name. For further information on public symbols, refer to sections later in this chapter.

The Linker and Slave Emulator use different terminology than the Assembler when referring to relocatable and absolute program segments; therefore, in Chapters 2 and 3 of this manual, RSEG is spelled CSECT and ASEG is spelled ASECT. In addition, public and external symbols are defined in the Assembler with the GLBL directive.

LINKER OUTPUT

The executable absolute object file generated by the Linker may be loaded into memory and executed by the Debugger. In addition, a memory map and reference list may be displayed or printed.

OPTIONS

Four Linker options are available to choose the desired type of input/output operation. Upon initial entry to the Linker, these options are displayed on the display screen. To select the desired options, type the appropriate letters as listed in Table 2-1. Options may be selected in any order either from the keyboard or from a command file.

Table 2-1. Linker Options

Option	Function
D	DELETE UNSPECIFIED SECTIONS Links only those sections which are specifically named in the Linker commands and deletes all other sections. If D is not selected, all sections from all input files will be linked, and the name and length of each section in each input file will be displayed. Sections named in input commands are positioned in memory as requested. All unnamed sections are positioned after the last named section.
S	INCLUDE SYMBOL TABLE Writes symbol tables from all input files to the output file. These tables, which relate public symbols to memory locations, are necessary for symbolic debugging. A symbol table may be placed in each relocatable object file by the Assembler or Compiler.
L	LIST TABLES ON CRT Displays the memory map and reference list.
A	APPEND DUPLICATE SECTIONS The normal operation of the Linker is to overlay multiple occurrences of a section. The Linker provides information in the link map when this occurs. This option forces all occurrences of duplicate sections to be concatenated rather than overlaid. This is particularly useful for compiler-generated assembly language programs, which may use section names in this manner.

COMMANDS

The Linker commands are entered to specify the following information:

1) the input file(s);

2) the output file;

3) the listing file (if any);

4) section placement in memory;

5) lists of section names in the order desired;

6) program execution entry point.

An example Linker display is shown in **Figure 2-1.** Options D and A were not selected. The next <RETURN> will cause Linker processing to begin.

```
========== Linker V5.2 ================================== FutureData ======


SPECIFY LINKER OPTIONS:
 (D) - DELETE UNSPECIFIED SECTIONS
 (S) - INCLUDE SYMBOL TABLE
 (L) - LIST TABLES ON CRT
 (A) - APPEND DUPLICATE SECTIONS
>LS
Input file options:  (select no more than one per file)
 /G - Globals only in symbol table
 /N - No symbol table at all
 /R - Reference only file (no data/symbols)

Input file:
>1:DEMO.R
Section      Length
CODE         0010
DATA         0BB8
Input file:
>
Output file:
>1:DEMO
Listing file or device:
>P:
Linker input:
>#ORG X'1000'
```

Figure 2-1. Example Linker Display

After the "Linker input:" prompt, the user may use the exclamation point character "!" to insert commentary which is ignored by the Linker but aids the user's memory/understanding. If the exclamation point is the first character of the input line, the whole line is ignored by the Linker; an exclamation point following an input command, on the same input line is treated by the Linker as the end-of-line character.

Each Linker command is defined below in the order of use.

Command	Explanation
input file-spec [input file-spec] . . .	After the Linker options have been entered, the Linker will prompt: INPUT FILE. Type the desired filename(s), one filename and <RETURN> per line. The Linker also supports three switches after an input file specification. The input files to the Linker are relocatable modules generated from the assemblers. They consist of text records, relocation information, and symbol tables. There is a symbol table for public symbols and one for local symbols. The three switches are used by the Linker to control the manner in which the input file is processed.
/R	Use the public symbol table of the file to resolve references in other files, but do not include any object records for this file.
/N	Do not include the symbol table of the file if the S option was chosen from the Linker menu. This is useful for creating smaller debugging modules that can be used with the debuggers.
/G	Similar to the /N switch in that it does not include local symbol tables for the file, but it does include the public symbol tables.
<RETURN>	Press <RETURN> after the last input file-spec has been entered.
[output file-spec]	The Linker prompts: Output file. Type the output file specification and press <RETURN>.
[listing file-spec]	The Linker prompts: Listing file. Type the listing file specification and press <RETURN>.
[#ORG absolute addr]	The Linker prompts: Linker input. Specify the absolute address of the first relocatable program section in the list following the #ORG statement.
[sect-a][,sect-b] [sect-c][,sect-d]...	Enter a list of section names. The sections are positioned in memory in the order named, starting at the address in the preceding #ORG or #SEG statement. If there is no preceding #ORG or #SEG directive, the starting address is zero.

Command	Explanation
[#ORG absolute addr] [sect-e]	Additional starting addresses, each followed by a list of sections, may be specified. The absolute address is decimal, hexadecimal, binary, or octal constant as described in Chapter 1. It is strongly recommended that the ORG addresses start at small values and only increase. This makes the memory map output much more readable.
[#SEG [long addr]]	This command specifies a 32-bit hexadecimal segment origin (similar to an ORG address but twice as long). If the long address parameter is missing, the segment origin defaults to the current load location pointer. There may be up to 15 of these commands per link. This directive is used in linking for execution by microprocessors which support more than 64K bytes of address space.
[#END [entry pt name]]	Enter a #END statement to begin Linker processing. entry point name is a public symbol specifying the program entry point. (Section names are automatically public.) If not specified, the program entry point is set to the address of the first section. After processing begins, press <BREAK> to halt Linker processing and view display output. Press <BREAK> again to continue. When processing is complete, the Linker displays the message: Function completed. Press <RETURN>. To begin Linker operation again, enter an input file-spec. To jump to another ADS system program, enter a J command.
[[#DEF] [entry pt name] [public symbol = absolute addr][SEG#]]	This command, followed by symbol definitions, allows the user to define values for external symbols which were not defined previously in some section but are referenced in some section. The DEF command, when used, must replace the #END command. The optional section number is a single hexadecimal digit separated by at least one blank from the address field.

LINKER SUB-COMMAND FILES

The user may invoke sub-command files within the Linker. If the input stream starts with a "." at any point at which the Linker expects input, the remainder of the string is taken as a file specification to be used as an alternate input stream. A sub-command file is distinguished by the "S" attribute. When the end of file marker is reached, the Linker returns to its previous input stream. The sub-command file may be invoked while command file processing is active and may be used any number of times. However, sub-command files cannot be nested.

The example in **Figure 2-1** is repeated in **Figure 2-2**, but in this case a sub-command file, LINK.FILES, is used. The contents of the file are:

```
1:DEMO.R

1:DEMO
P:
```

Figure 2-2 gives the resulting Linker display when run with the sub-command file. The line "Linker input:" is generated from the previous input stream, either via the keyboard or a command file. The lines may not include the command file commands such as n for parameter substitution, L and K.

```
============Linker V5.2=====================================FutureData=======

Specify Linker options:
  (D)  Delete unspecified sections
  (S)  Include symbol table
  (L)  List tables on CRT
  (A)  Append duplicate sections
>LS
Input file options:  (select no more than one per file)
 /G - Globals only in symbol table
 /N - No symbol table at all
 /R - Reference only file (no data/symbols)

Input file:
>.1:LINK.FILES
>1:DEMO.R
Output file:
>1:DEMO
Listing file or device:
>P:
Linker input:
>#ORG X'1000'
```

Figure 2-2. Example Linker Display Sub-Command Files

MEMORY MAP OUTPUT

#ORG commands may cause an CSECT to overlap a previously specified CSECT or the ASECT.

An example memory map display is shown in **Figure 2-3.** The memory map lists, for each section:

1) Its start address in memory (if the #ORG commands were entered with their operands in ascending order, the listed memory locations will also be in ascending order);

2) Its name;

3) The name of the file from which the section was taken;

4) Its length in hexadecimal.

```
============ Linker V5.2 ================================== FutureData ======

Addr          Section        File           Length

1000          PROGRAM        SKELETON.R     059F
159F          IOCODE         EXEC80.R       0028
157C          DEVICE         DEVICE.R       0005
15CC          INCODE         INPUT80.R      03BD
1989          KBCODE         KB80.R         0077
1A00          TVCODE         TV80.R         032D
1D2D          NIO            NET80.R        0565
2292          STACK          SKELETON.R     0064
22F6          DIO            DISK80.R       09CA
2CC0          LPCODE         LP80.R         0360
D600          WADATA         WORK.R         0200
D800          CRT            TV80.R         07D0

============ Linker V5.2 ================================== FutureData ======

File          Section   Addr      Length
SKELETON.R    STACK     2292      0064
              PROGRAM   1000      059F
```

Figure 2-3. Example Memory Map Output

If a section was deleted (option D), its name will appear in the memory map and the address will be flagged with a "D". If an section is overwritten by another section, the overwritten section will be flagged with an "O" in the memory map.

```
=========== Linker V5.2 ================================== FutureData ======

Addr          Section        File           Length

0080          INTER$         INIT86.R       0034
0000-1        INIT$          INIT86.R       0027
0027-1        INIT$          RARIT86.R      0006 A
002D-1        INIT$          CRT86.R        0005 A
0032-1        INIT$          TERM86.R       0007 A
0039-1        RUNTIME        MATH86.R       04C6
04FF-1        RUNTIME        STRING86.R     0187 A
0686-1        RUNTIME        SET86.R        0080 A
0706-1        RUNTIME        LARIT86.R      015F A
0865-1        RUNTIME        RARIT86.R      0674 A
0ED9-1        RUNTIME        MISC86.R       006B A
0F44-1        RUNTIME        CRT86.R        005B A
0000-2        ROM$           TMP.R          004D
004D-2        ROM$           MATH86.R       00AD A
00FA-2        ROM$           SET86.R        0060 A
015A-2        ROM$           RARIT86.R      0004 A
015E-2        ROM$           PASCALIO.R     001A A
0178-2        ROM$           REALIO.R       0035 A
01AD-2        RAM$           INIT86.R       0038
01E5-2        RAM$           RARIT86.R      0046 A
022B-2        RAM$           PASCALIO.R     0002 A
022D-2        RAM$           CRT86.R        07D2 A
09FF-2        HEAP$          MISC86.R       0002
3FFE-2        STK$           MISC86.R       0000
0000-3        CODE$          TMP.R          010B
09A4-3        PASCALIO       REALIO.R       02B1 A
010B-3        PASCALIO       PASCALIO.R     0899


SEG Address
 00 00000000
 01 1F054FCA
 02 CD000000
 03 041F047E
```

Figure 2-3a. Example Of Segmented Memory Map Output

Note: Every address in this output is followed by a hyphen and the segment number in which it resides.

REFERENCE LIST OUTPUT

An example Linker reference list display is shown in Figure 2-4. The display lists, for each relocatable file specified in the link:

1) Its file name;

2) For each section loaded from that file:

   a. Its section name;

   b. Its starting address in hexadecimal;

   c. Its length in hexadecimal.

3) For each public symbol in the section:

   a. The symbolic name;

   b. The memory address corresponding to that name.

```
========== Linker V5.2 ================================== FutureData ======

File          Section       Addr     Length
SKELETON.R    STACK         2292     0064
              PROGRAM       1000     059F

              Publics
              PROGRAM       1000          STACK     2292

File          Section       Addr     Length
EXEC80.R      IOCODE        159F     0028

              Publics
              IOCODE        159F          IOEXEC    159F

File          Section       Addr     Length
DEVICE.R      DEVICE        157C     0005
```

Figure 2-4. Example Linker Reference List Display

```
========== Linker V5.2 ================================== FutureData ======

File        Section  Addr     Length
INIT86.R    INTER$   0080-0   0034
            INIT$    0000-1   0027
            RAM$     01AD-2   0038

            Publics
            INIT$    0000-0   INTER$   0080-0   L$0      01AD-D   LASTATE  01E3-3
            RAM$     01AD-D

File        Section  Addr     Length
TMP.R       CODE$    0000-3   010B
            ROM$     0000-2   004D

            Publics
            CODE$    0000-0   M$$$$    0000-0   PACON    0000-0   ROM$     0000-0

File        Section  Addr     Length
MATH86.R    RUNTIME  0039-1   04C6
            ROM$     004D-2   00AD

            Publics
            ARCTAN   03FC-C   COS      0166-6   EXP      01A3-3   LN       0357-7
            RUNTIME  0039-9   SIN      0066-6   SQRT     025A-A

File        Section  Addr     Length
STRING86.R  RUNTIME  04FF-1   0187

            Publics
            CMP$O    05C5-5   CMPS$    05A4-4   PA2ST$   063F-F   PACON$   04FF-F
            SBSTR$   0556-6   STR2P$   0609-9   STR2ST$  0663-3   STRCON$  0529-9

File        Section  Addr     Length
SET86.R     RUNTIME  0686-1   0080
            ROM$     00FA-2   0060

            Publics
            FTBL$    011A-A   IN$01    06DE-E   SETBIT$  0686-6   TBL$     013A-A
            TTBL$    00FA-A

File        Section  Addr     Length
LARIT86.R   RUNTIME  0706-1   015F

            Publics
            DDIV$    0758-8   DMOD$    0848-8   DMUL$    0706-6
```

Figure 2-4a. Example Segmented Linker Reference List Display

LIST OF MESSAGES

**DELETED SECTION REFERENCE IN (filename)

An section which is present in an input file, but is not included in the output file, is needed to resolve address references.

(label) **DUPLICATE PUBLIC IN (filename)

The first occurrence of a public symbol is used for address references. All additional definitions are flagged as errors.

NOT A RELOCATABLE FILE

The Linker input file was not created by the Assembler or Compiler or the file's "R" attribute was changed with the Manager. Assign the "R" attribute to the file.

PARM ERR ... RESPECIFY

There is a syntax error in the Linker input or the specified section name was not found.

**RELOCATION ERROR IN (filename),record # xxxxH

Relocation record number xxxx of the input file was not correctly built. First, try to reassemble; else, notify your Service Representative.

**SEQUENCE ERROR IN (filename),record # xxxxH

The records in the named file are not in proper order. Reassemble or recompile.

**SYMBOL TABLE NOT FOUND IN (filename)

The symbol table was not included when the program was assembled or compiled. Reassemble or recompile, or simply ignore.

**TABLE OVERFLOW

The Linker needs more memory space.

(label) **UNRESOLVED REFERENCE IN (filename)

The external reference was not found in the public symbol table.

**PUB REC LNG ERROR IN (filename),record # xxxxH

Public symbol record number xxxx of the input file was not correctly built. First, try to reassemble; else, notify your Service Representative.

**REL REC LNG ERROR IN (filename),record # xxxxH

Relocation record number xxxx of the input file was not correctly built. First, try to reassemble; else, notify your Service Representative.

**EXT REC LNG ERROR IN (filename),record # xxxxH

External symbol record number xxxx of the input file was not correctly built. First, try to reassemble; else, notify your Service Representative.

**SYM REC LNG ERROR IN (filename),record # xxxxH

Record number xxxx of the input file was not correctly built. First, try to reassemble; else, notify your Service Representative.

**REL REC LNG ZERO IN (filename),record # xxxxH

Relocation record number xxxx of the input file was not correctly built. First, try to reassemble; else, notify your Service Representative.

DEBUGGING

INTRODUCTION

The 2300 Series Advanced Development System (ADS) is a software development tool providing users with editing, compiling, assembling, and linking capabilities. Files are maintained by a File Manager and can be used with a variety of devices, including floppy disk, hard disk, printer, and Memory Expansion Unit (MEU).

Once the software is developed, users must have a way to test the software on the actual hardware it was designed for and still be able to make modifications and corrections. The target system often does not have the facilities to allow a user to perform those tests. Its input-output structure may not be suited for gathering relevant statistics about the program. Programmers must write time-consuming diagnostic programs into the main program for debugging. These diagnostics may alter the behavior of a real-time system enough so that it fails to perform when the diagnostics are removed. The target system may have no facilities for isolating small sections of code (through trace or breakpoint functions, for example) so that their behavior may be examined. There must be some sort of interface between the ADS and the target system to provide these facilities.

The Slave Emulator Unit is such an interface. It is a hardware device that emulates the real-time aspects of the target system. It is connected to the target system via a connector that is exactly the same size, with the same number of pins, as the target processor. Users simply remove the target processor from its socket and substitute the Slave Emulator connector.

The Slave Emulator is connected to the ADS via an RS-232 serial port. The ADS interprets user commands to the Slave Emulator and transmits simple control instructions to the Slave Emulator for execution. The Slave Emulator transmits raw data to the ADS for format and display. The ADS acts as a user interface and a display station. The Slave Emulator acts as a control unit for the target processor.

The Advanced Development System (ADS) coupled with the Slave Emulator Unit provides a powerful system for developing and testing both hardware and software in a microprocessor-based system. The Emulator is based around the actual target processor to be imitated or emulated (for example, the Z80 microprocessor), and a sophisticated control system is imposed over it to provide the facilities required of a software development tool. Only the processor is emulated, allowing the actual hardware of the target system to be used during software development.

When users are ready to test or debug the software, they invoke the Slave Emulator Debugger and its special features.

- The ADS/Slave Emulator combination provides a display which can be split into multiple windows to allow viewing of different areas of memory simultaneously in hex/ASCII or disassembled format (including user labels), or Logic Analyzer trace data. Each window in the display can track a microprocessor register.

- Standard debugging features include symbolic and arithmetic expression evaluation.

- Flexible memory mapping capabilities partition the Emulator memory into user-defined blocks for RAM and ROM simulation. The entire target processor address space is available to the user. Blocks of simulation memory can be write-protected for RAM and ROM simulation. Write-protection is provided in multiples of 256-byte blocks. Mapping functions allow any 8K or 32K block of simulation memory to be mapped anywhere within the microprocessor's full address space, except when using 64K X 8-bit dynamic simulation memory. This memory is designed to simulate memory throughout the address space of typical 8-bit processors.

- Z80 processor interrupts can be invoked via the keyboard for target system interrupt routine testing.

- Help displays are available with a single keystroke, to provide explanations of functions, parameters, and syntax.

- Four breakpoints with nesting and complex breaking conditions are available. Conditions include memory or I/O address ranges, data values, a halt/snap mode, breakpoint counts and four external lines for breakpoint qualification. The snapshot mode displays the status of the CPU and memory contents at breakpoints by momentarily halting the processor.

- Microprocessor control lines can be selectively enabled or disabled from the target processor under user control. Programs may be executed at full speed or single-stepped. Full-speed execution is a real-time operation up to 6MHz.

- Up to four Slave Emulators may be attached to one ADS, allowing the user to emulate multiple target processors.

This chapter contains detailed descriptions of each Z80 Slave Emulator command. Individual command descriptions identify additional features.

CRT DISPLAY

Five main elements form the CRT display:

LOGO LINE

There are two kinds of logo lines displayed: the Executive logo line and the Debugger logo line. The Executive logo line appears as follows:

```
======= Emulator Executive VX.X ============================ GenRad DSD =====
```

X.X refers to the software version used. The Emulator Executive logo line is processor-independent.

When the user switches to a particular Slave Emulator Debugger, the logo line displayed becomes processor-dependent and appears as follows:

```
======= XXXXX Emulator VX.X/ROM VX.X ======================= GenRad DSD =====

      PROCESSOR      SOFTWARE     FIRMWARE
       TYPE          VERSION      VERSION
```

For example:

```
======== Z80 Emulator V2.4/ROM V2.4 ======================== GenRad DSD =====
```

If the software version is not compatible with the firmware version, the Slave Emulator will not operate. Memory and register contents will not display any data in most cases. Please refer to Table 3-1 to determine compatible software and firmware for all released versions of the Slave Emulator.

Table 3-1. Software/Firmware Matches

SOFTWARE	FIRMWARE
V2.4	V2.4
V2.3	V2.2
V2.2	V2.1

The logo line disappears when the user enters the first keystroke.

COMMAND LINE

Commands are input on the top line of the screen. A blinking cursor indicates the position of the next character to be input.

MESSAGE LINE

Target system state changes and certain error conditions are reported on the message line, which is the line below the command line.

A new status message is displayed in double intensity, with no Emulator identification. If there are Emulators daisy-chained together, the status message is displayed in double intensity, if the status applies to the current Emulator, or in reverse video if it applies to any other Emulator on the chain.

An old status message is displayed in normal intensity, with the message centered and surrounded by asterisks. If there are Emulators daisy-chained together, the message appears on the left side of the screen in normal intensity, with Emulator identification.

REGISTERS AND STACK

The next portion of the display shows the current register contents and the top eight words contained in the user stack.

MEMORY DATA

The remainder of the screen can be formatted in one to four separate data windows. One of these windows is defined as "current". A line of reverse video highlighting the current address line designates the current window. Non-current windows display only one address, not the entire line, in reverse video. Depressing <TAB> redefines the current window clockwise on the display. Commands affecting the memory contents are executed at the address indicated in the current window reverse video line.

Users may select any of eight screen maps (numbered 0-7) for viewing simulation or target system memory. They represent combinations of one to four independent windows. Refer to the SCreen map command for details.

Any window may be set to display either disassembled instructions or hexadecimal and ASCII data. If the Slave Logic Analyzer is present, special display modes may be selected to view bus data. See the WIndow mode command for details.

Figure 3-1 shows these elements in a typical Z80 display. The command line is indicated by the cursor. The remainder of the screen is divided into two independent windows below the register data. Window A (top) displays a disassembled program. This is also the current window, as indicated by the window's long reverse video line. Window B (bottom) displays the contents of memory in hexadecimal and ASCII representation. The current address in Window B is indicated by the block in reverse video. The status of the emulated processor is displayed on the message line; in this case, ****** Halted ******.

```
========= Z80 Emulator V2.4 /ROM 2.4 ==================== GenRad DSD ==========

                                 **** Halted ****
AF  FEA8        A'F' FFFF        PC   0000       I    00                  Stack
BC  DEEA        B'C' FFFF        SP   083D                      + 1,0  005D    9,8  DA26
DE  DFDF        D'E' FFFF        IX   FFFF       R    27          3,2  39E7    B,A  FD74
HL  DE34        H'L' FFFF        IY   FFFF    IFF2    00          5,4  951B    D,C  FDCA
Flags    CPH                     Flags'    CNPHZS                7,6  9DA2    F,E  DFAD

Tracking PC
0000  80           ADD    A,B
0001  12           LD     (DE),A
0002  D49547       CALL   NC,4795
0005  73           LD     (HL),E
0006  17           RLA
0007  D5           PUSH   DE
0008  80           ADD    A,B

FFD0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  |................|
FFE0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  |................|
FFF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  |................|
0000  80 12 D4 95 47 73 17 D5  80 B3 0C 00 0C 00 0C 00  |....Gs..........|
0010  0C 00 0C 00 0C 00 0C 00  0C 00 0C 00 0C 00 0C 00  |................|
0020  0C 00 0C 00 0C 00 0C 00  0C 00 0C 00 0C 00 0C 00  |................|
0030  0C 00 0C 00 0C 00 0C 00  0C 00 0C 00 0C 00 0C 00  |................|
```

Figure 3-1. Typical Z80 CRT Display

HELP DISPLAYS

Help displays explain the operation of Slave Emulator commands. Entering <?> or the <HELP> function key on a blank command line produces the general list of commands shown in **Figure 3-2.** Please refer to Figure 3-7 to determine the location of <HELP>, since the keycaps may not be marked.

Entering <?> or <HELP> on the command line after any individual command is typed produces a help display explaining the specific operation of that command.

switch display to emulator (0-8)	display
screen map (0-7)	<down arrow> (move forward 1 line)
<tab> (assign new current window)	<up arrow> (move back 1 line)
offset base addr =	<right arrow> (move forward 1 byte)
address space	<left arrow> (move back 1 byte)
window mode	+ (move forward 1 page)
edit memory map (Target/Simulation)	- (move back 1 page)
store	find
set register	execute
breakpoint (0-3)	<step>
reset breakpoint (0-3)	restart
clear all breakpoints	halt
qualify trace	interrupt
mode	enable control lines
load file	check for empty PROM
write	program PROM
assign test memory address	get data from PROM
transcribe keystrokes	verify data in PROM
call command file	?
; (display comment)	jump to system component

Figure 3-2. List Of Commands Help Display

COMMAND COMPLETION AND SYNTAX

GENERAL CONVENTIONS

The following general conventions apply to command input:

1. All input can be entered in either upper- or lowercase. Case is significant only when entering character strings for the "Store" and "Find" commands.

2. Keywords are recognized automatically. When the user enters enough of a keyword to uniquely identify it, the Slave Emulator software appends the remaining text of that keyword to the user's input on the command line and positions the cursor after the full text.

3. Keywords include the following:

   Command names
   Breakpoint and Logic Analyzer parameter names
   Breakpoint and Logic Analyzer mode names
   Target address space attributes
   Interrupt type names and related keyword parameters
   Target control line names

4. <TAB> or <CTRL-I> within command input indicates that a default value is to be supplied. The software formats this default on the command line as if the user had entered it and positions the cursor after the default text. Individual commands may take additional action, such as asking for confirmation. <TAB> is ignored where a default value is not meaningful.

5. The separators allowed between multiple fields within a single command are <SPACE> and <,>. Spaces which precede any field are ignored.

6. Command input normally must be terminated by <RETURN> before the specified action occurs. This termination may take the form of a response to a confirmation request, so that in some cases "Y" is also an appropriate termination.

7. Whenever a command's action may be destructive in any way if an error is entered in its text, that command must be confirmed before it is executed.

FILENAME SYNTAX

Filename syntax recognizes a unit specification field consisting of a device type character followed by a unit number and delimited by a colon. Either the device type or the unit number may be omitted, with the corresponding default being "all device types" or "all units". The possible combinations are:

M:	Access MEU (any installed units).
M*:	Access MEU (any installed units).
Mn:	Access MEU units n, where n = 0-3.
D:	Access local floppy disk (any installed units).
D*:	Access local floppy disk (any installed units).
Dn:	Access local floppy disk unit n, where n = 0-3.
*:	Access any unit, either MEU or local floppy.
n:	Access unit n, either MEU or local floppy, where n = 0-3.
P:	Access local printer. (Not applicable for Load command.)

When a filename begins with "P:", the Slave Emulator software immediately asks for confirmation. Other device types require that the user enter the filename string which identifies the file in an MEU or floppy disk directory.

If the user enters more than one unit specification, the last one entered is the one used. The user may not backspace over a unit specification once it has been entered.

There are two ways to enter a filename. The user may type the ASCII string for the filename, followed by <RETURN>. This indicates that the filename is precisely that string. Filename parsing checks to see if the file exists on the selected set of devices, then asks for confirmation in one of the following two ways:

1. If the file exists, the parsing routines prompt "[Old file]". If this name exists on more than one device, a copy on the MEU takes precedence over one on a floppy disk, and one on a low-numbered unit takes precedence over one on a high-numbered unit.

2. If the file does not exist, the parsing routines prompt "[New file]". They create the file on the unit with the most free space among the selected devices, if the user gives confirmation.

The user may also type a string of characters corresponding to the beginning of the filename, followed by a space. If this string matches the beginning of precisely one filename in the selected devices's directories, filename parsing completes the full filename on the command line and asks for confirmation with the prompt "[Old file]". If the string matches no file, or multiple files, filename parsing displays an appropriate message after the last input for several seconds, then looks for further input.

<TAB> completes portions of a filename in the same way that <SPACE> completes the entire filename. <TAB>-induced filename recognition is useful for names of the form <field1>"."<field2>. For example, a directory contains the following files:

```
MODULE.SRC
MODULE.REL
MODULE.OBJ
```

If the user enters a string such as "MOD" followed by <TAB>, filename parsing appends "ULE" and waits for further input, provided no other files begin with "MOD". At this point, typing "SRC"<RETURN>, "S"<SPACE>, or "S"<TAB> selects file "MODULE.SRC".

SLAVE EMULATOR COMMANDS

Slave Emulator commands consist of English language verbs which may be followed by operands. When the user enters enough characters of the verb to uniquely identify it, the Slave Emulator Debugger completes it by appending the remaining characters.

For example, enter the command:

SWitch display to emulator (0-8)

Only the letters SW must be entered to uniquely identify that command. The command characters to be entered by the user are shown in uppercase. They are shown throughout the manual in uppercase to distinguish between user input and system-supplied input. These characters may be entered in either upper-or lowercase. (See SEt dialog modes command for upper-and lowercase options).

Unless otherwise indicated, all commands must be followed by <RETURN> to be entered.

COMMAND LINE EDITING

<BACKSPACE> or <CTRL-H> may be used to delete one command line character at a time. Only that part of the command entered by the operator may be deleted. If a character has already been accepted and the command completed, the character cannot be deleted. <CAN> must be used to cancel the current command.

THE SLAVE EMULATOR EXECUTIVE

The Slave Emulator Executive is the entry point for the Slave Emulation system. Enter the command

JS<RETURN>

to invoke the Slave Emulator Executive. The display for the Slave Emulator Executive shown in **Figure 3-3** appears on the screen. The status of each Slave Emulator attached to the ADS is displayed. When the Slave Emulator is reset, status is as follows:

1. The target processor displays the message "Initializing", then "Halted".
2. The target system control lines are disabled.
3. The internal memory mapping specifies that the entire physical address space is mapped to simulation memory.
4. Write access is allowed and simulation memory blocks are mapped to consecutive block addresses starting at location 0.
5. Four commands, SWitch Emulator, SEt Dialog Mode, SPecify screen write options, and Display Comment, are now allowed, as illustrated in **Figure 3-3.**

```
========= Emulator Executive V2.4 ======================= GenRad DSD =========

Emulator 1        Z80v1 Halted

Emulator Executive commands are:

          Switch display to emulator (0-8)
          Set dialog modes
          Specify screen write options
          ;(display comment)
```

Figure 3-3. Slave Emulator Executive Display

Most of the individual commands in the Executive produces a help display when <?> or <HELP> is entered. The Executive commands are described in the following paragraphs.

SET DIALOG MODES

Users can choose whether to allow upper-and lowercase or to convert all alphabetic characters to uppercase with the SEt Dialog Modes command. The user can also define <RETURN> to signify either YES or NO when the system requires command confirmation. Most Slave Emulator commands require a confirmation to execute.

Upper-and lowercase entry and "<RETURN> to confirm means yes" are the default values for this command.

This command can only be accessed from the Slave Emulator Executive.

This command sets modes to control dialog functions.

First select a mode, then specify the value for that mode.

-"k" selects keyboard input mode:

    -"a" specifies "keyboard input allows lower case"

    -"f" specifies "keyboard input folds lower case": All lower case characters will be folded into upper case.

-"r" selects return-to-confirm mode:

    -"y" specifies "return to confirm means yes": Execute the command or subcommand which prompted "[Confirm]".

    -"n" specifies "return to confirm means no": Abort the command or subcommand which prompted "[Confirm]".

Figure 3-4. Set Dialog Modes Help Display

SWITCH DISPLAY TO EMULATOR (0-8)

This command transfers control from the Slave Emulator Executive to the specified Slave Emulator Debugger and displays the state of the target system on the screen. At this point all other Debugging commands are functional. To return to the Emulator Executive, use the command "SWitch display to emulator (0-8)", with "0" as the Emulator number, or <RESET>.

Please note that <RESET> resets all Slave Emulators attached to the ADS.

SPECIFY SCREEN WRITE OPTIONS

Users may write the current screen image into a file in UDOS source format. The file is any output file and may reside on the MEU, local floppy, or local printer.

The filename and the options for filtering special characters out of the data in the screen image must be specified. If the file is being written to the printer, the option "printable characters only" (P) is assumed, and no filtering options are prompted for. Files written with either the "printable characters only" or "no attribute characters" (N) filtering options can be displayed with the UDOS Editor. The UDOS Manager can dump those files written with the "all characters included" (A) option in hexadecimal format.

The command initializes the file so that the first screen image is written at the beginning of the file. After the file has been initialized, using the <F1> function key appends the current screen image to the file. This key is operational only if the user has completed the "specify screen write options" command since last executing or resetting the Slave Emulator software.

This command specifies where and how to write a copy of the screen image in response to function key F1.

Respond to the "Filename:" prompt with the name of the file to write.

Respond to the "Filtering option" prompt with the 1st character of:

-Printable characters: Write bytes valued 20H-7FH, filter out special FutureData characters and display attribute characters.

-No attribute characters: Write bytes valued 00H-7FH, filter out display attribute characters.

-All characters: Write all bytes (00H-FFH)

Writing to P: forces the filtering option to "Printable characters".

Figure 3-5. Specify Screen Write Options Help Display

;(DISPLAY COMMENT)

The Display Comment command allows users to make notations in command files. A command code of ";" from the keyboard specifies that further input from the keyboard or command file is to be echoed as usual, but the input is otherwise ignored.

"; (display comment)" echoes up to 79 characters of commentary text on the command line. This text is normally terminated by return.

Figure 3-6. Display Comment Help Display

START-UP PROCEDURE

The Slave Emulator requires the following ADS configuration:

1. The ADS must be a Z80-based system.
2. The ADS must contain 64K of memory in any configuration.
3. The ADS must have the UDOS boot PROM installed on the Z80 CPU. The boot message must read "UDOS Bootstrap", V1.3 or later.
4. The ADS must have a local disk drive or MEU for Slave Emulator software.

The following is a suggested sequence of events for installation and initiation of Slave Emulation. Information regarding hardware set-up is detailed in the **Slave Emulator Hardware Reference Manual** (2302-5003-00).

STEP 1 Remove the MPIO card from the ADS and strap it for Slave Emulator operation. Strapping should be as follows:

2 3 4 6 7 8 11 12 6S for standalone ADS

2 3 4 6 7 8 11 13 6S for Cluster Network ADS

Return the MPIO card to the ADS.

STEP 2 Connect the supplied RS-232 cable from the Slave Emulator, Port 1, to the ADS rear connector panel, Serial 1. If more than one Slave Emulator is being installed on the same ADS, the Slave Emulators must be daisy-chained as follows:

a. Port 2 of Slave Emulator 1 to Port 1 of Slave Emulator 2.

b. Port 2 of Slave Emulator 2 to Port 1 of Slave Emulator 3.

c. Port 2 of Slave Emulator 3 to Port 1 of Slave Emulator 4.

A maximum of four Slave Emulators may be attached to one ADS.

STEP 3 Connect the Emulator probe to the appropriate jacks on the rear panel of the Slave Emulator. **Note that the jacks are keyed to prevent incorrect installation. Do not force the connection.**

STEP 4 Insure that the Slave Emulator power switch is off. Connect the power cord to the Slave Emulator.

STEP 5 Power up the ADS station and disk drive. Insert the Slave Emulator system diskette into any drive. (The Slave Emulator software can be read from any drive, as long as all necessary files are located on the same diskette.)

STEP 6 Load the Slave Emulator Executive and Debugger by typing JS<RETURN>. The disk drive or the MEU accesses the program and overlays. The display on the CRT should resemble the display below:

```
========= Emulator Executive V2.4 ======================= GenRad DSD =========

Emulator Executive commands are:

        Switch display to emulator (0-8)
        Set dialog modes
        Specify screen write options
        ;(display comment)
```

STEP 7 Turn on the Slave Emulator power switch.

STEP 8 After power is applied, the drive containing the Slave Emulator software will be accessed for several seconds, (or for a longer period of time if daisy-chained Emulators are initializing simultaneously), before the following display appears:

```
========= Emulator Executive V2.4 ======================= GenRad DSD =========

Emulator 1          Z80v1     Initializing

Emulator Executive commands are:

        Switch display to emulator (0-8)
        Set dialog modes
        Specify screen write options
        ;(display comment)
```

STEP 9 Key in SW1<RETURN> to display status for Emulator 1. The Emulator Debugger screen should now be displayed. Refer to **Figure 3-1** for a sample screen display.

SLAVE EMULATOR SYSTEM FILES

The Slave Emulator system diskette contains the following files:

SLAVEM	Resident portion of the Slave Emulator software.
(SENRCOM)	Non-resident command routine.
(SEHELP)	Text of help displays.
(SEAPDM)	Processor-dependent code and tables used by the ADS.
(SESPDM)	Processor-dependent code and tables used by the Slave Emulator.
(SYM1)	Symbol table for Emulator 1.

Note that there is only one symbol table file allocated (SYM1). If there are Slave Emulators daisy-chained together (up to a maximum of four), additional symbol table files, one for each Emulator, must be generated. Use the File Manager to create a file (SYMn) for each Slave Emulator, where n is the Slave Emulator number.

SAMPLE COMMAND SEQUENCE

The following is a sample command sequence to initiate the Slave Emulation process:

JS	Jump to the Slave Emulator Executive. This is the beginning of Slave Emulator operation.
SW1	Switch to emulator (in this case, Emulator 1).
SC	Screen map selection. Provides a selection of eight screen maps for viewing program data.
WI	Window mode command. Data may be displayed in any combination of symbolic or hex format, and may be addressed as absolute or offset. Slave Logic Analyzer displays are also set with this command to display cycle or waveform data.
ED	Edit memory map substitutes Slave Emulator memory for target system memory, or assigns address ranges for Slave Emulator simulation memory.
EN	Enable control lines. This command displays available control line names and allows the user to enable or disable control signals generated by the target system.
L	Load program. This command loads a file from the MEU or a local floppy disk.
B	Set breakpoint. Up to four hardware breakpoints may be set with this command.
EX	Execute the program.
ST	Store the specified expression or string beginning at the current location.
H	Halt. Stop the target system and display the current register values.
CL	Clear all breakpoints.
W	Write to the disk.

COMMANDS

Each command available on the 2302 Slave Emulator system is described in this section of the manual. The commands are organized alphabetically. A complete list of commands is contained in **Table 3-2.** Underlined uppercase letters indicate required user input.

Table 3-2. Slave Emulator Debugger Commands

COMMAND	FUNCTION
ADdress Space	Not applicable to the Z80 processor.
ASsign Test Memory Address	Not applicable to the Z80 processor.
Breakpoint(0-3)	Set a breakpoint.
CAll Command File	Invoke a command file which functions as a command subroutine.
CHeck for empty PROM	Check for empty PROM.
CLear all breakpoints	Clear all breakpoints.
Display	Position data in current window.
;(Display comment)	Allow user to insert comments in command files.
EDit Memory Map	Specify configurations of simulation memory and target address space.
ENable Control Lines	Enable or disable control lines from target system.

Table 3-2. (Continued)

COMMAND	FUNCTION
EXecute	Begin program execution at the program counter.
Find	Search for data in memory.
Get data from PROM	Move data from PROM to Emulator simulation or target memory.
Halt	Stop target processor.
Interrupt	Simulate interrupt to the target processor.
Jump to system component	Invoke an operating system component or a command file.
Load	Load object file into simulation or target memory.
Mode	Allow user to specify modes for breakpoint and Logic Analyzer operation.
OFfset Base Addr	Define display offset base address (effective only when window mode is set for offset addresses).
Program PROM	Program a PROM.
Qualify trace	Specify Logic Analyzer trace qualification.
RESEt breakpoint(0-3)	Clear a breakpoint.

Table 3-2. (Concluded)

COMMAND	FUNCTION
RESTart	Restart the target processor by asserting a hardware reset.
SCreen Map	Specify screen map (number and configuration of windows).
SEt Dialog Modes	Modify upper/lowercase, yes/no <RETURN> confirmation. May be used only within the Slave Emulator Executive.
SEt Register	Modify register contents.
STore	Store data into memory.
SWitch Display to Emulator (0-8)	Activate the individual Emulator Debugger.
Transcribe Keystrokes	Copy all keystrokes into a source file suitable for later use as a command file
Verify data in PROM	Compare PROM with simulation or target memory.
WIndow Mode	Set window mode to format in symbolic (disassembled) or hexadecimal format, to display cycle, waveform, or execution data, and to use either offset or absolute addresses.
WRite	Write simulation or target memory contents to an object file.

Table 3-3. Special Keys

SPECIAL KEYS	PURPOSE
<BACKSPACE>	Edit command line.
<CAN>	Cancel a command or a sub-command.
<LOAD>	Return to the boot loader.
<RESET>	Reset both the ADS and the Slave Emulator system.
THE FOLLOWING KEYS ARE COMMANDS:	
<TAB>	Redefine current window or supply default parameters.
<RETURN>	Update display to current status.
<STEP>	Execute Z80 instruction pointed to by the Program Counter.
<?>	Produce help display.
Up, Down, Left, Right Arrows	Position data in current display window.
<+>,<->	Position data in current display window.
<HELP>	Produce help display.
F1 (function key)	Write display function key; causes the current screen image to be appended to the file specified in the last "Specify screen write options" command.
;	Comment command; allows command files to supply prompts to the user.

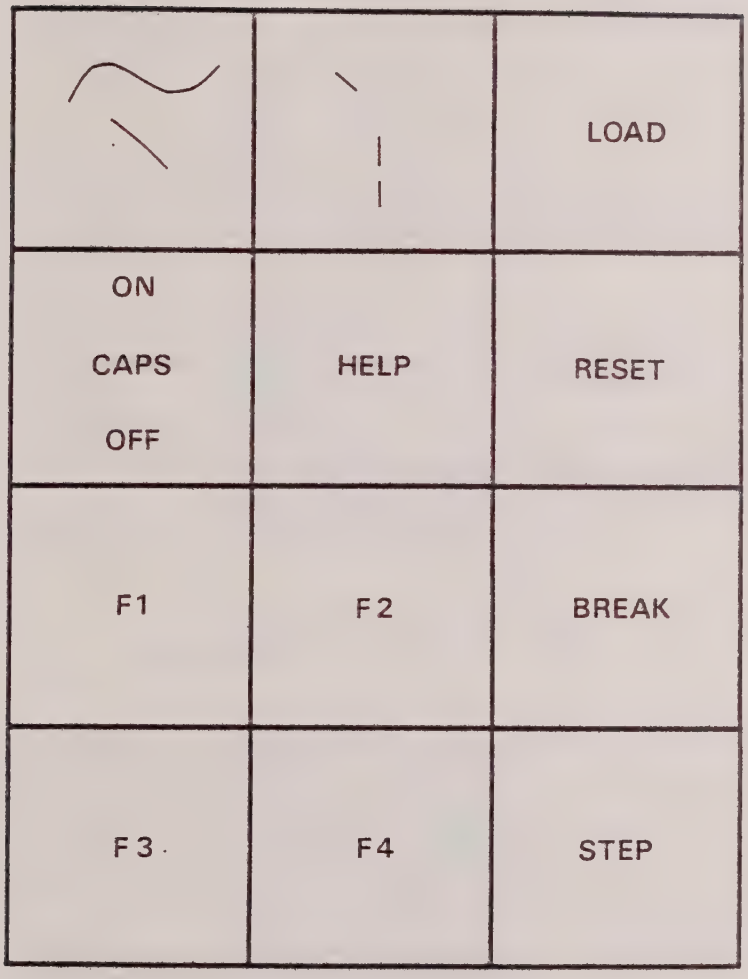


Figure 3-7. Special Keys

BREAKPOINT COMMAND

```
Breakpoint (0-3) n<RETURN>
                 n,
```

FUNCTION:

The Breakpoint command is used to both arm breakpoint n and to set its parameters. Four hardware breakpoints (0-3) are available.

OPERATION:

n specifies the breakpoint to be armed or to have parameters set.

<RETURN> arms breakpoint n and sets its parameters to defaults: break on instruction execution from current location of current window. To disarm the breakpoint, use the RESEt or CLear commands.

"," specifies that breakpoint parameters are to be set or changed. The "," activates the following subcommands:

OPTION	EXPLANATION
Count = n	1≤n≤32,767. Breakpoint parameters must be satisfied n times before the breakpoint is triggered. If the value entered is not in this range, a message **Invalid count value** flashes on the command line and aborts the subcommand. The old count value is not altered.
Address = <abs-addr>	<abs-addr> is any expression representing a 16-bit absolute address. If this expression is a hexadecimal number, "X" may be used in place of individual hexadecimal digits to indicate that any value matches. Breakpoints may break on a value less than or equal to, or greater than or equal to, the specified value.
Data =<data>	<data> is any expression representing an 8-bit data value. If this expression is a hexadecimal number, "X" may be used in place of individual hexadecimal digits to indicate that any value matches.

OPTION		EXPLANATION
External lines = bin 4		bin 4 (4 binary digits) is the status of the four external probes. X is used for "don't care" bits. The rightmost digit is line E0, the leftmost digit is line E3.
Instruction/data =	I	I specifies break on instruction execution.
	D	D specifies break on data access.
	X	X specifies "don't care".
Read/write =	R	R specifies break on read access.
	W	W specifies break on write access.
	X	X specifies break on "don't care".
Memory/IO =	M	M specifies break on memory address space access.
	I	I specifies break on I/O address space access.
	X	X specifies 'don't care'.
Halt/snap =	H	H specifies that the target processor be halted when all breakpoint conditions are satisfied.
	S	S specifies that the display be updated with the target status when breakpoint conditions are satisfied and that the target processor resume executing when the update is complete. The S (snap) option stops the emulator from several hundred microseconds to several milliseconds, depending on the target system's clock rate and other timing parameters, except when using scope 0 with "Run after trace" mode set. Refer to Qualify Trace command.
		The Slave Emulator processes a "halt" regardless of how short the interval is between snapshots. If the user issues commands other than "halt" while the interval between snapshots is extremely short, these commands are queued for processing AFTER the next halt command.

Note that if a breakpoint is armed and is executing, subsequent changes made by the user in breakpoint parameters will not take effect until the target processor is halted and an "Execute" command is entered.

Break parameter:

```
breakpoint (0-3) n  delim
    count = dec
    address = [<= or >=] abs
    data = data
    external lines = bin4
    instruction/data = I, D, or X
    read/write = R, W, or X
    memory/io = M, I, or X
    halt/snap = H or S
```

```
Breakpoint 1
      Count = 1
      Address = FFFF
      Data = XX
      External lines = XXXX
      Instruction/data = I
      Read/write = R
      Memory/io = M
      Halt/snap = H
```

```
 delim   , to edit parameters, other to set execution breakpoint
n        Breakpoint number:  0-3
 dec     Decimal repetition count:  1 to 32767
 abs     Expression or up to 6 hex digits, X for "don't care" digit
 data    Expression or up to 4 hex digits, X for "don't care" digit
 bin4    4 binary digits, X for "don't care" digit
```

Figure 3-8. Breakpoint Help And Status Display

CALL COMMAND FILE

CAll command file (filename)

FUNCTION:

This command invokes a command file which functions as a command subroutine. It does not require jumping to another system component.

OPERATION:

The Call Command File command first prompts for a filename, which the user enters using the standard Slave Emulator filename dialog conventions. The user is then prompted for parameters; entering only <RETURN> indicates that no parameter values will be supplied to the command file. Otherwise, the input is a sequence of strings separated by commas and terminated by <RETURN>.

Call command file (filename)

This function invokes a command file without jumping out of the Slave Emulator software.

Respond to the "call command file(filename)" prompt with the name of a UDOS source file containing valid Slave Emulator command input.

Respond to the "Parameters" prompt with:

-- <RETURN> if the command file does not require parameters.

-- Text of parameters, terminated by <RETURN>. Multiple parameters are separated by commas.

Figure 3-9. Call Command File Help Display

CHECK FOR EMPTY PROM

CHeck for empty PROM

FUNCTION:

The CHeck command verifies that a PROM is completely erased.

OPERATION:

The command initially prompts for PROM type. The following PROM types are valid: 2704, 2708, 2716, 2516, 2732, 2532, 2758. <TAB> defaults initially to 2716. If the PROM type is changed, the new value becomes the next default.

```
                n
PROM length:
              <TAB>
```

n is an expression that specifies the number of bytes to be programmed beginning at the start of the PROM. n is a hexadecimal value.

<TAB> defaults to program the complete PROM.

```
                          n
Interleaving factor:
                        <RETURN>
```

n specifies the interleaving factor between bytes stored in memory.

Interleaving factor is not meaningful to the Check command. It is prompted for consistency with dialog for the other PROM commands. <TAB> initially defaults to 1. The last value set for any PROM command is the default value.

For further information, please refer to the EPROM PROGRAMMER USER'S MANUAL, (2300-5035-00).

```
Check for empty PROM PROM type:

Check for empty PROM
    PROM type:              expr      Type of PROM chip
                            TAB       Default PROM type (initially 2716)

    Length                  expr      Length of data in PROM
                            TAB       Default length:  Chip capacity

    Interleaving factor     expr      Offset between PROM bytes in target
                                      memory

                            TAB       Default interleaving factor:  1

PROM chips supported:
2704, 2708, 2758, 2716, 2516, 2732, 2532
```

```
expr     Expression formed with + or - operators, symbols and hex numbers
         (symbols have the form "#symbol)
```

Figure 3-10. Check Command Help Display

CLEAR BREAKPOINTS COMMAND

CLear all breakpoints [Confirm]

FUNCTION:

This command clears all previously set breakpoints.

OPERATION:

The CLear command disarms all breakpoints. This command disables the hardware breakpoints and initializes all breakpoint parameters to default values. There is no help display for this command.

DISPLAY COMMAND

Display

FUNCTION:

The Display command selects the area of memory to be displayed in the current window. addr may be an absolute hexadecimal address or one of the other forms described below.

OPERATION:

```
Display #symbol
        or
Display #module:symbol
```

symbol is defined in the original assembly language program. Display is positioned to the address associated with that symbol. A "#" character precedes the symbol name.

Symbols may be qualified by module names by entering the command as "#module:symbol".

Display ±expr

expr is evaluated as a 16-bit value and is added or subtracted to the current display address. The value of expr is saved for subsequent Display ± commands.

An expression consists of one or more terms separated by + or - signs. Each term may be a hexadecimal value or a symbolic reference.

Display @reg

reg is a register name as shown in the Slave Emulator display. This command displays the address pointed to by the contents of the named register in the current window.

The display now tracks the contents of the register(s) specified. The message "Tracking reg" is displayed in the top line of the window.

```
Display BReakpoint(0-3) n

Positions current window to address specified by breakpoint n.  If the
breakpoint address contains "don't care" bits, they are replaced by "0" bits.


Display

                expr            address = expr
                + expr          address = current address + expr
                - expr          address = current address - expr
Display         @ reg           address = contents of reg relocated by default
                                sreg
                @ sreg:reg      address = contents of reg relocated by sreg
                @ sreg          address = 0 relocated by sreg
                Breakpoint n    address = address of breakpoint n

Display +expr or -expr saves expr as page size for + or - commands
```

```
expr     Expression formed with + or - operators, symbols and hex numbers
         (symbols have the form "#symbol")
reg      Any register in the target processor
sreg     Any register in the target processor used to specify
         the base of a memory segment
n        Breakpoint number:  0-3
```

Figure 3-11. Display Command Help Display

EDIT MEMORY MAP COMMAND

EDit memory map (Target/Simulation) [T / S] [Confirm]

FUNCTION:

The Edit Memory Map command allows the user to use all or part of a target system's address space. It defines the emulation environment of the target processor in terms of memory access behavior.

Simulation memory can be used in place of target system memory for code generation and testing in the early stages of a project while the hardware is being developed. As hardware becomes available, Simulation memory can be replaced with target system memory, or used as ROM or RAM simulation in the final testing stages of a product.

The memory mapping facility determines whether to use simulation or target memory. The target processor's address space can be divided into several areas, each with different response attributes.

OPERATION:

Edit Target Memory

Initially, the entire address space of the target processor is mapped to direct all memory accesses internally to the Slave Emulator, and to allow write accesses. This is the default and reset condition for this command.

The Target (T) option controls the direction of memory accesses and the attribute of specified areas of the target processor's address space. The command prompts for several parameters and uses these to build a table of descriptions of memory mapped areas. This table is sent to the Slave Emulator for processing when the user completes the command.

Target Option Subcommands:

Block start: addr
[<RETURN>]

addr is the hexadecimal address of a block of Slave Emulator memory to be assigned to the target system. This address is a multiple of X'100'.

If <RETURN> is entered instead of an address, the command is terminated. See **Figure 3-12** for the EDit memory map (Target) help display.

Block length: n

n is the hexadecimal length of the block of memory to be mapped. n must be in multiples of 256 (X'100') bytes.

After the start and length of the block are entered, the command prompts for the attributes of the block. These attributes describe where the memory accesses are to be directed and the types of accesses allowed.

External/Simulation

E directs all memory accesses to the defined block to the target system.

S directs all memory accesses to the defined block to be internal to the Slave Emulator and does not pass them to the target system.

The Write Allow/Prohibit attribute is the final prompt. It controls write accesses to Simulation memory for ROM/PROM simulation. Setting the attribute to Prohibit Write prevents writing to Simulation memory. This memory protection feature is meant to work only in conjunction with Simulation memory. No control over target system memory behavior is possible.

Allows/Prohibits write access

A allows a write to the defined block. P prohibits writes to the defined block.

Figure 3-13 shows the map summary displayed on the screen when using the Target option.

```
Block start:

Target address space map selects attributes for blocks of target addresses.

Command dialog identifies a block by start address and length,
then assigns attributes to it.

    Start address identifies block's first byte, must be multiple of 256.
    --Enter hexadecimal absolute address or <return> to quit.

    Length identifies extent of block, must be multiple of 256.
    --Enter length in hexadecimal.

    Memory attribute (Simulation/External) selects simulation memory in SECU
    or external memory in target system.

    Write protect attribute (Allow/Prohibit) allows or prohibits
    write accesses to this block.
```

Figure 3-12. Edit Memory Map (Target) Help Display

```
Attributes:  Simulation/External, Allow/Prohibit writes ("EA" block not shown)
          Start    End   Length  Attr     Start    End   Length  Attr
          0000     FFFF  10000    SA
```

Figure 3-13. Initial Edit Memory Map (Target) Display

Edit Simulation Memory

Upon initialization or reset, the Slave Emulator maps all installed Simulation memory to contiguous addresses starting at location zero in the target processor's address space.

The Edit Simulation Memory command is used to position physical blocks of Simulation memory within the address space of the target processor. These physical blocks are of fixed size, dependent upon the type of memory board installed in the Slave Emulator. This command must be used with the Edit Target Memory command to set up the emulation environment for the target procesor. Care must be taken to first download any code to be executed from this area before attributing it as "Write Prohibit", if the area is to be used in ROM/PROM simulation. Code can not be downloaded into a write-protected area of Simulation memory.

Edit Simulation Memory Subcommands:

```
Block number:   n
              [<RETURN>]
```

n is the block number which identifies each installed block of simulation memory. Blocks are sequence-numbered.

<RETURN> terminates the command. See **Figure 3-14** for the Edit Memory Map (Simulation) help display.

```
Block start:    addr
              [<RETURN>]
```

addr is the absolute address assigned to the first byte of this block of simulation memory. addr must be a multiple of the block length. Blocks of Simulation memory cannot be positioned so as to overlap each other. The command dialog will refuse input that attempts to overlap blocks.

Entering <RETURN> without an address will unmap the block. Unmapping a block of simulation memory prevents its use.

Figure 3-15 shows the map summary displayed on the screen when using the simulation option command.

```
Block number:


Simulation memory map assigns target addresses to blocks of simulation memory.

    Block number identifies each installed block of simulation memory
    --Enter decimal integer to select a block, or <return> to quit.

    Start address is target address at which block's first byte is mapped
    --Enter hexadecimal absolute address, or <return> to unmap block.

Block start address must be multiple of block length and blocks cannot overlap.
This map does not affect whether target system accesses go to simulation
memory; only target address space map selects access to simulation memory.
```

Figure 3-14. Edit Memory Map (Simulation) Help Display

```
Block number:

 Block    Start    End     Length      Block    Start    End     Length

  00      0000     1FFF    2000         01      2000     3FFF    2000
  02      4000     5FFF    2000         03      6000     7FFF    2000
```

Figure 3-15. 32K System Initial Edit Memory Map (Simulation) Display

ENABLE CONTROL LINES COMMAND

ENable control lines: X

FUNCTION:

The ENable control lines command determines to which target system's control lines the Slave Emulator will respond.

OPERATION:

X represents one of the following:

NONE	no lines active (default)
ALL	all lines active
RESET	reset line
NMI	non-maskable interrupt
INT	interrupt request line
BUS REQUEST	bus request
WAIT	wait line
CLOCK	clock

Enter enough characters to uniquely identify the control line. The software completes the control line name and asks for confirmation. In some cases, additional information is prompted for. The "-" character in front of the control line name disables that line or block of lines. "-ALL" disables all lines. Reverse video indicates lines "mapped external". <RETURN> terminates the command.

ENable control line

Enter name from list below to enable control line(s)
Enter -name from list below to disable control line(s)
Enter RETURN to terminate command

None
All
Reset
Nmi
Int
Bus Request
Wait
Clock

Figure 3-16. Enable Control Lines Display

EXECUTE COMMAND

EXecute [Confirm]

FUNCTION:

The EXecute command begins user program execution.

OPERATION:

The current contents of the Program Counter (PC), as shown in the Register Display, determine the memory address at which execution begins.

Execution is halted by one of the following methods:

Encountering an active breakpoint.
Using the Halt command.
Using the Load command.
Using the STEP key.

EXecute [Confirm]

Execute causes the target system to resume execution.

Execution resumes at the address contained in the target processor's program counter register, relocated by a segmentation register if appropriate.

Figure 3-17. Execute Command Help Display

FIND COMMAND

```
        <data>[<data>]...
Find    <'string'>
         <TAB>
```

FUNCTION:

All forms of the Find command search memory for the occurrence of data or strings.

OPERATION:

<data> is a 1- to 8-digit hexadecimal expression, but only the two low-order digits are significant. Each data value corresponds to an 8-bit byte. <string> is a series of ASCII characters enclosed in single quotes ('). <TAB> defaults to the previous data or string defined in the last Find or STore Data command.

The Find command allows the user to enter up to 16 bytes of data to the command line. When the user enters the last byte that the command line can accommodate, the Debugger supplies the byte string's delimiter.

Find

```
        expr expr...
Find    'ascii string'
        <TAB>
```

expr expr... is a string of expressions representing byte values

'ascii string' may contain any ascii characters except its delimiter "'"

<TAB> defaults to the operand of the last store or find command

Find begins searching one byte after the current location in the current window;
It searches to the end of:

- --The target address space, if a window shows absolute addresses
- --Window's 64K segment, if window shows offset addresses

Figure 3-18. Find Command Help Display

GET DATA FROM PROM COMMAND

Get data from PROM

FUNCTION:

The Get command loads the information contained in a PROM into the target address space beginning at the current address of the current window. This command may also be used to fill memory (memory is filled with FFFF's) if the PROM programmer module is not connected.

OPERATION:

PROM type:

The following PROM types are valid: 2704, 2708, 2758, 2716, 2516, 2732, 2532. <TAB> initially defaults to 2716. The default is the last PROM type entered.

PROM length: n / <TAB>

n specifies the length of data to be transferred from PROM. <TAB> defaults to PROM size.

Interleaving factor: n / <TAB>

n specifies the interleaving factor used to move bytes of PROM to memory in the target address space. <TAB> initially defaults to 1. If n = 2, sequential bytes may be programmed alternately in multiple PROMs.

For further information, please refer to the EPROM PROGRAMMER USER'S MANUAL (2300-5035-00).

```
Get data from PROM PROM type:

Get data from PROM
    PROM type:              expr      Type of PROM chip
                            TAB       Default PROM type (initially 2716)

    Length                  expr      Length of data in PROM
                            TAB       Default length:  Chip capacity

    Interleaving factor     expr      Offset between PROM bytes in target
                                      memory

                            TAB       Default interleaving factor:  1

PROM chips supported:
2704, 2708, 2758, 2716, 2516, 2732, 2532
    ** This command executes for several minutes for typical PROM types **
```

```
expr    Expression formed with + or - operators, symbols and hex numbers
        (symbols have the form "#symbol)
```

Figure 3-19. Get Data From PROM Help Display

HALT COMMAND

Halt [Confirm]

FUNCTION:

The Halt command suspends execution on the target processor.

OPERATION:

The Halt occurs when the command is confirmed with <RETURN>.

Halt [Confirm]

Halt suspends execution on the target processor

Figure 3-20. Halt Command Help Display

INTERRUPT COMMAND

Interrupt

FUNCTION:

The Interrupt command simulates an interrupt to the target processor.

OPERATION:

Interrupts available on the Z80 Slave Emulator are:

```
Mode 0 (RST, JP or CALL instr)
Restart at address 8 x n
Int mode 1 (CALL 38H)
NMI (CALL 66H)
Vectored (Int mode 2)
```

The interrupt is issued when <RETURN> is entered.

The user must enter only enough characters to uniquely identify the desired interrupt. In several cases, additional information is prompted for.

```
Interrupt

        Mode 0 (RST, JP or CALL instr)
        Restart at address 8 x n
        Int mode 1 (CALL 38H)
        NMI (CALL 66H)
        Vectored (Int mode 2)
```

Figure 3-21. Interrupt Command Help Display

JUMP TO SYSTEM COMPONENT

Jump to system component <character>

FUNCTION:

<character> is the first character of a Z-attributed system file name. Entering "JS" loads the Slave Emulator Executive from the bootstrap or from another system program, such as the Manager or Editor.

OPERATION:

The Jump to System Component command loads an executable file (a Z-attributed file) to the system. Examples of system component files are the Manager, the Editor, the Linker, the Assembler, and the Slave Emulator Executive. Typing a "J" plus the first letter of the system filename loads the file.

For further information, please refer to the **UDOS REFERENCE MANUAL, Chapter 2, UDOS Overview (2301-5002).**

LOAD FILE

Load file filename

FUNCTION:

The Load file command halts the target processor, if running, searches the available storage devices for the specified filename, and loads the file into the target processor address space.

OPERATION:

If the filename is not prefixed by a device type code or unit number, the Slave Emulator software searches for the file first on the MEU, then on all locally installed floppy disk drives, beginning with drive 0.

The Load command issues the following prompts:

```
                          addr
Load segment base addr =
                         <TAB>
```

addr specifies the offset address into which to load the file.

<TAB> or <RETURN> defaults to 0 initially or to the previous base address. This parameter does not relocate address references within the loader file. Relocation is performed only by the Linker.

This command automatically distinguishes between Linker I and Linker II files by the contents of the first record. The above dialog is required by Linker I files. Linker II files require entry of the filename only. Information on segmentation and address space use is contained within the object file itself.

The Load command sets the program counter to the start address specified by the Assembler/Linker or the Write command, whichever created that file. If the filename specified exists on more than storage device and the user does not explicitly specify a unit or drive number, the MEU and then the floppy disk drives starting with unit 0 will be searched and the first occurrence of that filename will be loaded.

If the specified filename is not found, the message ****no such file**** will be briefly displayed and the system will await the next command.

Load file

Load file filename loads the named file into the target address space.

Additional input for Linker I object files:

"address space = same as current window's [Confirm]" prompt appears if the target processor supports multiple address spaces. Confirm to load into the current window's address space, or abort and use "address space" command to switch address spaces.

The "Load segment base address = expr" subcommand specifies the absolute base address for a 64K segment which the file will be loaded into.
<TAB> defaults to last value used (0 initially).

---

filename	Name of an object file
expr	Expression formed with + or - operators, symbols and hex numbers (symbols have the form "#symbol")

Figure 3-22. Load File Help Display

MODE COMMAND

Mode parameter:

FUNCTION:

The Mode command controls the Slave Logic Analyzer and breakpoint operation. This command's dialog is similar to that of the Breakpoint command and its display appears as shown below when all modes are set to their initial defaults.

OPERATION:

Typing the command automatically brings up the help display shown in Figure 3-23.

AND/OR breaks mode specifies how different breakpoints interact:

The "OR" setting indicates that all enabled breakpoints function independently.

The "AND" setting indicates that all enabled breakpoints function together. Initially, only the lowest-numbered active breakpoint is armed; when its conditions are satisfied, it arms the next higher-numbered breakpoint, rather than causing an actual break. The break finally occurs when the highest-numbered active breakpoint's conditions are satisfied.

Snapshot scope indicates how much data the Slave Emulator updates when the target system reaches a snapshot breakpoint.

0: Nothing	The Slave Emulator does not interrupt the target system and does not report register or memory contents to the ADS. This is useful only when the breakpoints are used to trigger the Logic Analyzer while the target system is running in full-speed emulation.
1: Register Contents	The Slave Emulator interrupts the target system for about 10 milliseconds to examine the current register contents and reports these to the ADS.

2:	Register Contents and Memory Displayed	The Slave Emulator interrupts the target system for about 100 milliseconds to examine the current register contents and check for altered data in memory areas whose contents are buffered in the ADS for the Debugger's display. Target processor execution restarts as soon as it has fetched new register and memory contents. If a previous set of values is queued for transmission to the ADS, but transmission has not started, the Slave Emulator discards the earlier register or memory contents and reports the most recent data.
3:	Snap All at Each Break	The target processor remains halted until all data reported by the snapshot has been transmitted to the ADS and the ADS has completely updated its display. This ensures that the display always shows consistent snapshot data and that each snapshot's results are displayed. Target processor execution is suspended for a period of time between 1/2 second to several seconds during data transmission and display updating.

In the display of current mode settings, this parameter appears as follows:

0 (Snap nothing)
1 (Snap regs only)
2 (Snap regs & mem)
3 (Snap all)

Enable/Disable Analyzer enables or disables Logic Analyzer operation. In this mode, reaching a breakpoint (either ORed or ANDed) supplies a Logic Analyzer trace trigger. Execution continues until the number of qualified cycles indicated by the trace qualifier's post-trigger delay have been traced.

When enabling the Logic Analyzer the Debugger issues the following information message on the message line:

****** Analyzer enabled, break 3 disabled ******

When disabling the Logic Analyzer the Debugger issues the following information message on the message line:

****** Analyzer disabled, break 3 enabled ******

The Slave Emulator hardware uses breakpoint 3 to specify the trace qualifier when the Logic Analyzer is enabled.

Run/Pause after trace specifies whether or not the target processor is to continue running while the Slave Emulator copies out the contents of the Logic Analyzer's trace buffer. This applies only when the triggering breakpoint specifies the "snap" option.

The Run setting allows the target processor to continue execution. If the snapshot scope mode is 0 this causes Logic Analyzer trace updates with absolutely no interruption of target system execution. Other settings of the snapshot scope mode delay the target system by the times quoted above.

The penalty for this mode is that breakpoints and Logic Analyzer tracing must remain disabled while the Slave Emulator copies the trace buffer, so that the target system may miss breakpoints. The duration of this critical period will be about 5-10 milliseconds.

The pause setting halts the target system while the Slave Emulator copies the Logic Analyzer trace buffer. This ensures that breakpoints cannot be missed and that all target system execution is traced, but it suspends target execution for about 10 milliseconds more than the delay needed to satisfy the breakpoint scope mode.

The Timer Units mode indicates what units the Logic Analyzer's event timer uses. The event timer is active only when Breakpoint 1 specifies a start condition and Breakpoint 2 specifies an end condition and the following modes are in effect:

```
AND/OR breaks = A
Scope = 0 (snap nothing)
Enable/disable analyzer = E
Run/pause after trace = R
```

Under these conditions the event timer begins clocking real time or bus cycles when the conditions for breakpoint 1 are satisfied, and stops when breakpoint 2 triggers the Logic Analyzer. Breakpoint 0 has no effect on the event timer.

The event timer uses a 24-bit counter in one of three modes:

Microseconds:	Counter increments once per microsecond, elapsed interval is reported to user in microseconds.
Nanoseconds:	Counter increments once per 100 nanoseconds, elapsed interval is reported to user in nanoseconds.
Bus cycles:	Counter increments once per target system bus cycle, elapsed interval is reported to user as bus cycle count.

The elapsed time or bus cycle counts observed by the event timer are reported in the ADS display's message area with the following message:

Running, elapsed time n units

In this message 'n' is the decimal event timer value and ' units ' is 'nanoseconds', 'microseconds', or 'bus cycles'.

"Default all mode parameters to initial values" sets all parameters to the standard value in effect when the Slave Emulator is reset. These settings are:

```
AND/OR breaks = 0
Scope = 3 (Snap regs only)
Enable/disable analyzer = D
Run/pause after trace =P
Timer units = microseconds
```

The ADS requires the user to confirm in order to accomplish this mode change.

"Set all mode parameters for event timing" sets the parameters to the values needed to activate the event timer, and sets the timer units parameter to microseconds. The user must confirm by depressing <RETURN> or "Y" to accomplish the mode change.

```
Mode parameter:

Mode                                  |Breakpoint/logic analyzer modes
                                      |
    and/or breaks = A or O            |    And/or breaks = O
    scope =  scope                    |    Scope = 3 (snap all at each break)
    enable/disable analyzer = E or D  |    Enable/disable analyzer = D
    run/pause after trace = R or P    |    Run/pause after trace = P
    timer units =  unit-code          |    Timer units = microseconds
                                      ------------------------------------
    Default all mode parameters to inital values.
    Set all mode parameters for event timing.

-------------------------------------------------------------------------

 scope        Scope of information updated at a snapshot breakpoint:
                   0 = Nothing
                   1 = Register contents
                   2 = Register contents and memory being displayed
                   3 = Registers and memory @ every snap
      **Logic Analyzer trace is independent of snapshot breakpoint scope
 unit-code    Units used by event timer:
                   Microseconds
                   Nanoseconds (minimum resolution = 100 nanoseconds)
                   Bus cycles

-------------------------------------------------------------------------
```

Figure 3-23. Mode Command Help Display

OFFSET BASE ADDRESS COMMAND

OFfset base addr= expr

FUNCTION:

The Offset Base Address command assigns the specified value to the window's absolute base address.

OPERATION:

When the window mode is set for offset addressing, all addresses are treated as displacements from this base address.

expr is an expression, evaluated as a 16-bit absolute base address.

Addresses shown in the display window are relative to the offset specified when window mode is 'offset addresses'. This function can be used to view a particular relocatable program segment (RSEG) relative to its start address, rather than relative to zero in memory. For example, if an RSEG is loaded at location X'1420', the user can enter "OFfset base addr = 1420". Then, by entering "Display 0", the current window shows location X'1420' absolute, but identifies it as location X'0'. In this way, addresses displayed on the screen will match the assembly listing.

OFfset base addr =

Offset base addr = expr

expr defines absolute start address of 64K segment to display
when window mode is set for offset addresses

---

expr	Expression formed with + or - operators, symbols and hex numbers (symbols have the form "#symbol")

Figure 3-24. Offset Base Address Help Display

PROGRAM PROM COMMAND

Program PROM

FUNCTION:

The Program command begins the PROM programming process using data beginning at the current address in the current window. <RETURN> continues the command.

OPERATION:

PROM type:

The following PROM types are valid: 2704, 2708, 2716, 2516, 2732, 2532, 2758. <TAB> initially defaults to 2716. The default PROM type is the last PROM type set by any PROM command.

PROM length: n <TAB>

n specifies the number of bytes to be programmed beginning at the start of the PROM. <TAB> defaults to program the complete PROM.

Interleaving factor: n <TAB>

n specifies the interleaving factor between bytes programmed from the target address space into the PROM. For example, this factor allows alternate bytes to be programmed into 8-bit PROMs to form 16-bit words. <TAB< initially defaults to 1.

For further information, please refer to the EPROM PROGRAMMER USER'S MANUAL (2300-5035-00).

```
Program PROM PROM type:

Program PROM
    PROM type:              expr      Type of PROM chip
                            TAB       Default PROM type (initially 2716)

    Length                  expr      Length of data in PROM
                            TAB       Default length:  Chip capacity

    Interleaving factor     expr      Offset between PROM bytes in target
                                      memory

                            TAB       Default interleaving factor:  1

PROM chips supported:
2704, 2708, 2758, 2716, 2516, 2732, 2532
    ** This command executes for several minutes for typical PROM types **
```

---

expr Expression formed with + or - operators, symbols and hex numbers (symbols have the form "#symbol)

Figure 3-25. Program PROM Help Display

QUALIFY TRACE

Qualify trace

FUNCTION:

The Qualify Trace command sets parameters to select bus cycles to be traced when the Analyzer is enabled. These parameters are identical to those used with breakpoints, but the meaning of a match condition is "trace this cycle" rather than "break execution". This command also sets the post-trigger delay, which indicates how many qualified bus cycles to trace after the triggering event occurs.

OPERATION:

Qualify trace functions exactly as the breakpoint command, except that the parameters it sets are those of the Logic Analyzer's trace qualifiers (Breakpoint 3). The following display appears on the screen when the command is entered.

```
qualify trace                          | Trace qualifier
    post-trigger delay =  dec          |     Post-trigger delay = 1
    address =  [<= or >=] abs          |     Address = XXXX
    data =  data                       |     Data = XX
    external lines =  bin4             |     External lines = XXXX
    instruction/data = I, D, or X      |     Instruction/data = X
    read/write = R, W, or X            |     Read/write = X
    memory/io = M, I, or X             |     Memory/io = X
```

```
dec          Decimal bus cycle count:  1-255
abs          Expression of up to 6 hex digits, X for "don't care" digit.
data         Expression of up to 4 hex digits, X for "don't care" digit.
bin4         4 binary digits, X for "don't care" digit.
```

Figure 3-26. Qualify Trace Help Display

Trace qualifier parameters are:

Post-trigger delay: Number of bus cycles to trace after a trigger (after a breakpoint). This is a decimal number 1-255, with the default being 1.

Address: Address pattern to trace. This is a hexadecimal number in which "X" in any digit indicates "don't care". The number of digits in this number depends on the width of the target processor's address bus. The default is entirely "don't care" digits.

Data: Data pattern to trace. This is a hexadecimal number in which "X" in any digit indicates "don't care". The number of digits in this number depends on the width of the target processor's data bus. The default is entirely "don't care" digits.

External lines: External line values to trace. This number consists of four binary digits in which "X" in any digit indicates "don't care". The default is entirely "don't care" digits.

Instruction/data:

- I — Trace only opcode-fetch cycles.
- D — Trace only data access cycles. Most processors do not distinguish between operand fetches and fetches for the second and subsequent bytes or words of instructions.
- X — Default; means trace both instruction fetches and data accesses.

Read/write:

- R — Trace only read cycles.
- W — Trace only write cycles.
- X — Trace both read and write cycles.

Memory/IO:

M	Trace only cycles which access memory.
I	Trace only cycles which access I/O ports.
X	Default; trace both memory and I/O accesses.

The trace qualifier enables tracing of all bus cycles when all of its parameters have the default "don't care" setting.

For further information regarding the Slave Logic Analyzer, please refer to the **SLAVE LOGIC ANALYZER MANUAL (2302-5012-00).**

RESET BREAKPOINT COMMAND

RESEt breakpoint (0-3) n

FUNCTION:

This command is used to disarm a previously set breakpoint.

OPERATION:

This command disables the specified hardware breakpoint.

There is no help display for this command.

RESTART COMMAND

RESTart

FUNCTION:

The RESTart command issues a hardware reset to the target processor.

OPERATION:

The reset is issued when <RETURN> key is entered. The message "Running" is displayed on the message line.

RESTart [Confirm]

Restart performs a hardware restart of the target system
by applying a reset control signal

Figure 3-27. Restart Command Help Display

SCREEN MAP COMMAND

SCreen map(0-7) n

FUNCTION:

The Screen Map command maps up to four display windows onto the screen, with windows assigned to quadrants of the display as shown below.

OPERATION:

Any of eight screen maps may be selected for viewing target processor memory. The desired map number is specified by the letter n, as shown in the help display (**Figure 3-28**). The default screen map is n = 2. Each window may display symbolic (disassembled), hexadecimal/ASCII memory data, or Logic Analyzer trace data in any of three formats.

Logic Analyzer trace data can only appear in a window which spans the full width of the screen. (See WIndow mode command). If the current window is the incorrect size to support Logic Analyzer data, an error message stating "XXX **Requires wide window**" will be displayed and the command aborted. This error message may be issued by both the Screen Map command and the Window Mode command.

SCreen map (0-7)

Screen map n maps up to four display windows onto the screen, with windows assigned to quadrants of the display as shown below.
Windows are labelled A - D.

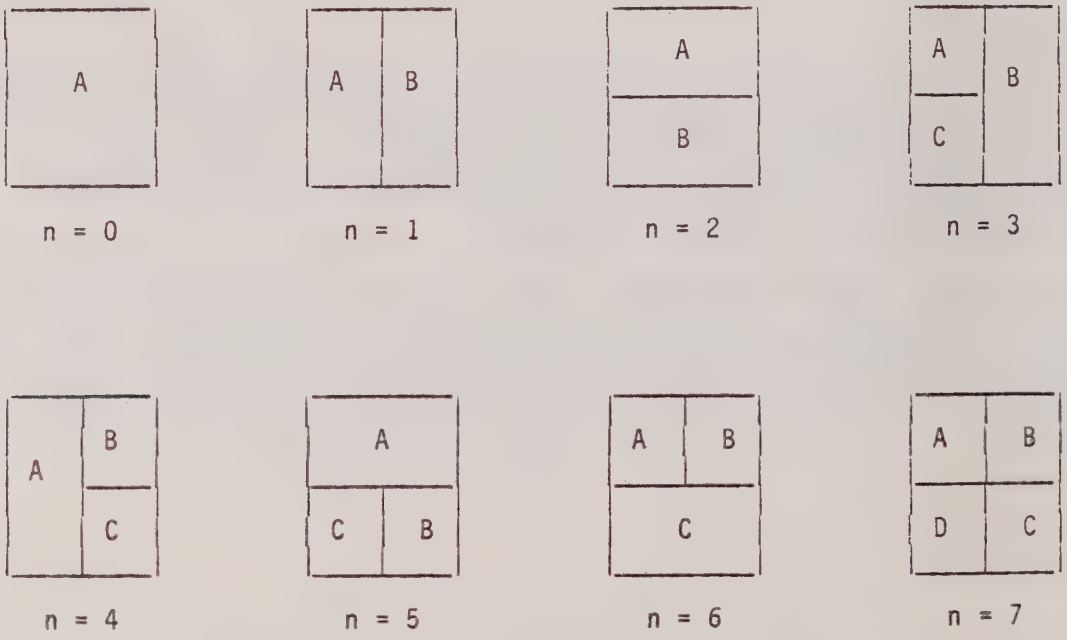


Figure 3-28. Screen Map Help Display

SET REGISTER COMMAND

SEt register =<reg>

FUNCTION:

The SEt Register command sets the contents of the specified register to 'data'.

OPERATION:

<reg> may be any Z80 register name shown at the top of the Debugger display, except the R register. <data> is one to four hexadecimal digits; a symbol or an expression. The number of significant bits depends upon the size of the register.

The SEt register command also allows the user to change the the target processor's flag or condition code values as if these data were a register.

The Z80 emulator displays the target registers in the following formats:

```
========= Z80 Emulator V2.4 /ROM 2.4 ==================== GenRad DSD ==========

                              **** Halted ****
AF  FEA8      A'F' FFFF       PC   0000      I    00                 Stack
BC  DEEA      B'C' FFFF       SP   083D                  + 1,0  005D    9,8  DA26
DE  DFDF      D'E' FFFF       IX   FFFF      R    27       3,2  39E7    B,A  FD74
HL  DE34      H'L' FFFF       IY   FFFF    IFF2   00       5,4  951B    D,C  FDCA
Flags    CPH                  Flags'    CNPHZS            7,6  9DA2    F,E  DFAD
```

Figure 3-29. Z80 Register Display

```
SEt register

-------------------------------------------------------------------------

Set register reg = expr

-------------------------------------------------------------------------

reg     Any register in the target processor
expr    Expression formed with + or - operators, symbols and hex numbers
        (symbols have the form "#symbol)
```

Figure 3-30. Set Register Help Display

STORE COMMAND

```
     <data>[<data>]...
STore <'string'>
      <TAB>
```

FUNCTION:

All forms of the STore command store data strings into memory.

OPERATION:

<data> is a 1- to 8-digit hexadecimal number (or symbol or expression). Only the low 8 bits of this value are significant. <'string'> is a series of ASCII characters enclosed in single quotes ('). <TAB> defaults to the previous data or string defined in the last Find or STore command.

The STore command does not allow the user to enter enough data to overflow the command line. When the user enters the last byte the line can accommodate, the Debugger supplies input of the byte string's delimiter. Confirmation is requested before entering.

The STore operation begins placing the data or string into memory at the current location in the current window.

```
              expr expr ...
Store         'ascii string'
              TAB
```

expr expr ... is a string of expressions representing byte values

'ascii string' may contain any ascii character except its delimiter, "'", and preemptive keystrokes, such as "?", <BACKSPACE>, <CAN>, etc.

TAB defaults to the operand of the last store or find command

---

```
expr          Expression formed with + or - operators, symbols and hex numbers
              (symbols have the form "#symbol")
```

Figure 3-31. Store Command Help Display

SWITCH DISPLAY TO EMULATOR (0-8)

SWitch display to emulator (0-8) n

FUNCTION:

The SWitch command switches from the Emulator Executive to the specified Slave Emulator and displays its status on the screen.

OPERATION:

If n = 0, control is returned to the Slave Emulator Executive. If n = any number from 1 to 8 inclusive, the specified Slave Emulator is switched to for interrogation.

All communication links are always active. The SWitch command switches control of the ADS display and keyboard to the process communicating with the specified emulator. All emulators may be simultaneously running a user program in the target system.

The control lines, memory maps and execution parameters for each emulator are unaffected when switching from one emulator to another. **Figure 3-33** displays the status of the target system after using the SWitch command. The emulator status may be any one of the following:

Halted
Running
Completed single step
Stopped @ breakpoint n
Stopped @ ANDed breakpoint
Running, snapshot @ breakpoint n
Running, snapshot @ ANDed breakpoint
Bus timeout
Target system check

```
Switch display to emulator (0-8) n

        n selects one of the attached emulators;
        0 returns control to the Emulator Executive
```

Figure 3-32. Switch Command Help Display

```
========= Z80 Emulator V2.4 /ROM 2.4 ==================== GenRad DSD ==========

                                **** Halted ****
AF  FEA8      A'F' FFFF      PC   0000      I    00                 Stack
BC  DEEA      B'C' FFFF      SP   083D                  + 1,0  005D    9,8  DA26
DE  DFDF      D'E' FFFF      IX   FFFF      R    27       3,2  39E7    B,A  FD74
HL  DE34      H'L' FFFF      IY   FFFF    IFF2   00       5,4  951B    D,C  FDCA
Flags   CPH                  Flags'   CNPHZS             7,6  9DA2    F,E  DFAD

Tracking PC
0000  80          ADD    A,B
0001  12          LD     (DE),A
0002  D49547      CALL   NC,4795
0005  73          LD     (HL),E
0006  17          RLA
0007  D5          PUSH   DE
0008  80          ADD    A,B

FFD0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  | ................ |
FFE0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  | ................ |
FFF0  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  | ................ |
0000  80 12 D4 95 47 73 17 D5  80 B3 0C 00 0C 00 0C 00  | ....Gs.......... |
0010  0C 00 0C 00 0C 00 0C 00  0C 00 0C 00 0C 00 0C 00  | ................ |
0020  0C 00 0C 00 0C 00 0C 00  0C 00 0C 00 0C 00 0C 00  | ................ |
0030  0C 00 0C 00 0C 00 0C 00  0C 00 0C 00 0C 00 0C 00  | ................ |
```

Figure 3-33. Typical Display After Switch Command

TRANSCRIBE KEYSTROKES

Transcribe keystrokes to filename:

FUNCTION:

This command copies all keystrokes into a source file for later use as a command file.

OPERATION:

When the user enters and confirms his filename selection, the ADS opens this file for write access. The end of file pointer is set to show that the file is empty.

The ADS then prompts:

Include prefix sequence? [Confirm]

If the user enters <RETURN> or "Y" to confirm this option, the ADS writes the following text at the beginning of the file:

```
JS
^L
SW1Y
```

This is a standard prefix for command files to be invoked by a Jump command. It implements the following three functions:

1. Jump to the Slave Emulator software. A side affect of this is that it resets all emulators attached to the ADS.

2. Wait for the user to enter <RETURN>. This ensures that the next command can operate properly if Emulator 1 is slow in resetting.

3. Switch to Emulator 1.

Command files to be invoked by the "Call command file" command would not use this prefix.

Transcription of keystrokes begins when the dialog is complete. It ends when the user enters <CTRL-E> or jumps to a different system component.

Transcribe keystrokes to filename:

This command initiates transcription of keystrokes to a UDOS source file, which may be used later as a command file.

Respond to the "Transcribe keystrokes to filename:" prompt with the name of the file to write.

Respond to the "Include prefix sequence? [Confirm]" prompt with <RETURN> or "Y" to include the prefix sequence, "N" or <CAN> to exclude it. This prefix sequence, appears as follows in the transcript file:

```
JS
^L
SW1
```

This sequence is used in command files to initiate Slave Emulator operation.

Entering <CTRL-E> at any time terminates keystroke transcription and closes the transcript file.

Figure 3-34. Transcribe Keystrokes Help Display

VERIFY DATA IN PROM COMMAND

Verify data in PROM

FUNCTION:

The Verify command compares the information contained in a PROM with that beginning at the current address of the current window in the target address space.

OPERATION:

PROM type:

The following PROM types are valid: 2704, 2708, 2716, 2516, 2732, 2532, 2758. <TAB> initially defaults to 2716. The default value is the last PROM type entered for any PROM command.

```
                n
PROM length:
              <TAB>
```

n specifies the number of bytes that were verified beginning at the start of the PROM. <TAB> defaults to verify the complete PROM.

```
                      n
Interleaving factor:
                    <TAB>
```

n specifies the interleaving factor between consecutive bytes in the target address space. If n = 2, sequential bytes may be programmed alternately in multiple PROMs. <TAB> initially defaults to 1. The default value is the last value entered.

For further information, please refer to the EPROM PROGRAMMER USER'S MANUAL (2300-5035-00).

```
Verify data in PROM PROM type:

Verify data in PROM
    PROM type:              expr      Type of PROM chip
                            TAB       Default PROM type (initially 2716)

    Length                  expr      Length of data in PROM
                            TAB       Default length:  Chip capacity

    Interleaving factor     expr      Offset between PROM bytes in target
                                      memory

                            TAB       Default interleaving factor:  1

PROM chips supported:
2704, 2708, 2758, 2716, 2516, 2732, 2532
```

```
expr    Expression formed with + or - operators, symbols and hex numbers
        (symbols have the form "#symbol)
```

Figure 3-35. Verify Data in PROM Help Display

WINDOW MODE COMMAND

WIndow mode parameter

FUNCTION:

The Window Mode command specifies the type of display to be selected for viewing emulator memory. Only the first letter of the mode is entered.

OPERATION:

The Window mode command affects the current window. The current window is indicated by a reverse video line spanning the window at the current address. Each window may be set individually by redefining the active window with the TAB key.

For further information regarding this command, please refer to the SLAVE LOGIC ANALYZER MANUAL (2302-5012-00).

WIndow mode

Window mode command operands:

Symbolic	Display memory data as disassembled symbolic instructions
Hex	Display memory data as hexadecimal bytes and ASCII characters
Cycle data	Display raw (minimally formatted) logic analyzer trace data
Waveform	Display signals as waveforms from logic analyzer trace data
Execution	Display execution trace from logic analyzer trace data
Absolute	Use absolute addresses, referencing entire target address space
Offset	Use 16-bit offset addresses, referencing a 64K segment beginning at the window's offset base address

** Absolute/offset mode applies to memory displays, not to analyzer traces

Figure 3-36. Window Mode Help Display

WRITE COMMAND

WRite from

FUNCTION:

The WRite command writes the contents of memory to an object file.

OPERATION:

address space name is requested only if the target processor allows address spaces. <expr> is the start address of the block to write. TAB defaults to 0. The default address appears at the tab position on the command line.

The second <expr> prompted for is the ending address of the block to write. <TAB> is the default end address, which is the highest target address.

```
                addr
start addr=
               <TAB>
```

addr is the start address of the program being written relative to the segment base address. <TAB> defaults to the start address specified in the last file loaded. The default address appears at the tab position on the command line.

```
write into file:    filename
```

If "[old file]" appears and is confirmed, the old file is overwritten. If "[new file]" appears and is confirmed, the file is created and written.

```
WRite from


Write
      address space     Address space name or <RETURN>
                        [requested only if target processor allows address
                        spaces]

      from              expr      Start address of block to write
                        TAB       Default start address:  0

      to                expr      End address of block to write
                        TAB       Default end address:  highest target address

      start addr =      expr      Execution start address
                        TAB       Default execution start addr from last file
                                  loaded

      into file         filename  Name of file to write

**Delimit end addr with "," to enter another block description
```

```
filename      Name of an object file
expr          Expression formed with + or - operators, symbols and hex numbers
              (symbols have the form "#symbol"
```

Figure 3-37. Write Command Help Display

SPECIAL KEYS

<BACKSPACE>	Deletes one character at a time from the command line.
<CAN>	Deletes the entry on the command line. Cancels the current command or subcommand.
<LOAD>	Returns control to the ADS terminal bootstrap. All boot level commands are allowed at this point.
<RESET>	Resets both the hardware and the software for the ADS and all Slave Emulators attached to it. Control of the display and keyboard is returned to the Emulator Executive.
<RETURN>	Multiple functions including entering a command; updating of the register, stack and memory displays to show the present status of the specified emulator; in some cases, terminating a command or a subcommand; unmapping a block of memory. Refer to specific commands for details.
<TAB>	Redefines the current display window when used as a command. The current window is indicated by a reverse video line across that window. All commands which affect memory begin at the address in the reverse video line of the current window. <TAB> also supplies a default value in many commands when used as an operand. <TAB> may be used in conjunction with the Find or STore commands, the PROM length subcommand and PROM type, PROM interleaving factor, Load command filename, Load offset, WRite block start addresses, WRite block end address, WRite execution start address, WRite filename, break parameters, etc. to invoke a default value. Refer to these commands individually for specific instructions.
<STEP>	Executes one user program instruction at a time. Execution begins at the address contained in the program counter. To allow the current window to track single step program execution, the window must be set to track the instruction pointer using the Display command.

<?> Provides help displays to explain operation of Slave Emulator commands. When entered on a blank command line, a list of all available commands is displayed. Any subsequent keystroke returns to the previous display. Refer to Figure 3-2.

When <?> is entered following a command on the command line, explanatory information specific to that command is displayed. Completion of the command or subcommand replaces the help display with an appropriate display.

<;> The semicolon allows the user to put comments in the command files. These are not executed, but merely echoed to the display.

F1 This is a special function key that appends the current screen image to the file specified in the last "Specify screen write options" command. It has no effect if the user has not completed this command since last jumping to the Slave Emulator software.

DISPLAY CONTROL KEYS

<↑> The up arrow backs up the current location of the current window one instruction for a symbolic window or one line for a hex window.

<↓> The down arrow advances the current location of the current window one instruction for a symbolic window or one line for a hex window.

<→> The right arrow advances the current location of the current window one byte.

<←> The left arrow backs up the current location of the current window one byte.

<+> The plus character moves forward one page in memory. Refer to the Display command for definition of page size.

<-> The minus character moves back one page in memory.

GenRad

DSD SERVICE LOCATIONS

UNITED STATES AND CANADA

CO	300 Baker Avenue Concord, MA 01742 617/369-4400 or 617/646-7400 TWX: 710-347-1051	Conn., Maine, Mass., N.H., R.I., VT.
CHO	1083 East State Parkway Schaumburg, IL 60195 312/843-5580 TWX: 910-291-1209	IL., Iowa, Minn., Ohio, MO., N.D., S.D., KY., Ind., Mich., W.PA., Wis., Neb.
DEN	13132 St. Paul Drive Thornton, CO 80241 303/457-9147 (Ans. Serv.)	Colo., Mont., Wyoming, N.M., Utah
DSD	5730 Buckingham Parkway Culver City, CA 90230 (Factory) 213/641-7200; TWX: 910-328-7202	Factory Field Support
DTX	1121 Rockingham, Suite 100, Richardson, TX 75080 214/234-3357; TWX: 910-867-4771	Ark., Kan., LA., Tex., Okla.
FLO	3751 Maguire Blvd Suite 170, Orlando, FL 32803 305/894-4303; TWX: 810-850-0270	Ala., GA., Fla., N.C., S.C., Tenn., Miss.
LAO	17631 Armstrong Avenue P.O. Box 19500,Irvine, CA 92714 714/540-9830; TWX: 910-595-1762	S.CA., Ariz., S.Nev.
NYO	22-08 Route 208 Fair Lawn, NJ 07410 201/797-8001 (NJ); 212/964-2722 (NY) TWX: 710-988-2205	Del., E.PA., N.J., N.Y.
SFO	2855 Bowers Avenue, Santa Clara, CA 95051 408/727-4400; TWX: 910-338-0291	N.CA., Idaho, N. Nev., Wash., Ore.
WO	1701 Research Blvd, Rockville, MD 20850 301/424-6224; TWX: 710-828-9783	MD., VA., W.VA., D.C.
GRC	307 Evans Avenue Toronto, Ontario, Canada M8Z 1K2 416/252-3395; TELEX: 06-967624	All Canada

SERVICE LOCATIONS

EUROPE

DENMARK	Mr. Steen Schulstad BLT Agenturer A/S GL. Koge Landevej 55 DK-2500 Valby	Telephone: (01) 16 11 00 Telex: 855-16279 BITEK DK
SWEDEN	Mr. Stig Svensson, General Mgr. Mr. Gosta Olsson, Sales Mgr. Lagercrantz Electronik AB, Box 48 S-194 21 Upplands Vasby	Telephone: 760 861 20 Telex: 854-11275
FINLAND	Mr. Erik Hilden, Director Oy Emmett Ab Meterorinkatu 3 C-D 02210 ESPOO 21, Finland	Telephone: 009358/0882044 Telex: 857-22216
NORWAY	Mr. T.S. Fjeldstad, Director Bergman Instrumentering AS P.O. Box 129 Veitvet Sven Oftedals Vei 10 Oslo 5, Norway	Telephone: 16 22 10 Telex: 856-17271
NETHERLANDS	Mr. Michael Houdijk C.N. Rood BV Cort Van de Lindenstraat 11-13 2280 Rijskijk	Telephone: 070 996 360 Telex: 844-31238
FRANCE	Mr. Alberto Franzetti, AMM 6 Avenue Du General DeGaulle 78150 Le Chesnay	Telephone: 945 91 13 Telex: 842-698376
SPAIN	Mr. Francisco Herbada, General Mgr. Mr. Miguel Angel Ferrando, Sales Mgr. Hispano Electronica S.A. Poligono Industrial Urtinsa Apartado Correos 48 Alcorcon, Madrid	Telephone: 619 41 08 Telex: 831-22404
SWITZERLAND	Mr. Peter Multhanner, Sales Mgr. Mr. Ernst Schmid, General Mgr. Mr. Hans Wagenmakers GenRad (Schweiz) AG Drahzugstrasse 18, Postfach 8032 Zurich, Switzerland	Telephone: 55 24 20 Telex: 845-53638
AUSTRIA	Mr. Gunther Graf Kontron Elektronik A-2345 Brunn A. Geb Industriestrasse B13 Vienna, Austria	Telephone: 02236/866310 Telex: 847-79337

ITALY	Mr. Germano Fanelli Celdis Italiana Cinisello Balsamo 20092 Via Fill Gracchi, 36	Telephone: 612 00 41 Telex: 843-334883
GERMANY	Mr. Guido Negele, President Kontron Messtechnik GMBH Breslauer Strasse 2 D-8057 Eching Munich, W. Germany	Telephone: 089 3 1901 217 Telex: 841-522122
ENGLAND	Mr. David Findlay Kontron, Ltd. Campfield Road St. Albans AL1 5JG England	Telephone: Telex: